FINAL REPORT

ADL PROTOTYPE

Design and Development of a 3D Simulation Component Repository Supporting ADL Training and Education Applications

soVoz, Inc.
37 Station Drive
Suite A
Princeton Jct., NJ 08550

Contract # N61339-02-C-0151
Period: 6/1/04 – 12/31/05
Date: 12/31/05

Table of Contents

TABLE OF CONTENTS

SECTION I

In recent years, simulation has been recognized as having great educational potential because of its ability to facilitate "learning by doing." As a result, numerous attempts have been made to integrate 3D simulation into existing ADL applications, focusing mainly on the interaction between the simulation and the Learning Management System (LMS). Most 3D simulation technology, however, is expensive, inflexible, and incompatible with SCORM's principles of "interoperability, accessibility, and reusability." Warfighters representing each of the Military Services are demanding simulations that provide a relevant, adaptable, and doctrinally accurate training capability that is responsive to the rapidly changing Contemporary Operating Environment (COE). Supplementary research is now required within the distance learning environment to make these advanced simulation technologies coexist with SCORM's reusable content model. A research platform consisting of authoring tools and a library of shareable simulation objects was developed in this project to aid in exploring the full potential of using interactive simulation in ADL training and education applications.

INTRODUCTION

This Prototype primarily supports creation of "Simulations to Enhance Instruction." It addresses how to quickly develop simulated scenarios/environments that support Advanced Distributed Learning initiatives and how to integrate reusable simulation components with existing SCORM-conformant instructional content. The soVoz simulation technology developed as part of an OSD Phase II SBIR effort was leveraged in the proposed Prototype and was used as a research platform for exploring these issues.

What was unique about this project, compared to the previous efforts, was the focus on the actual process of creating interactive simulation scenarios for ADL applications using a component-based approach. Given a particular application domain, design guidelines were established for determining the set of required reusable simulation components; a prototype of a content library containing application-specific collections of reusable components was developed, organized by type and function to enable high-level authoring of simulation scenarios by instructional designers and subject matter experts; and SCORM-conformant simulation content, created using components accessed from the library, was used to validate the efficacy of the component-based approach.

This final report contains descriptions of the three simulation scenarios developed in this project to support the training of Emergency Response Teams (ERT) responding to hazardous materials (HazMat) / weapons of mass destruction (WMD) incidents. Each of the simulation scenarios developed was intended to provide a "hands-on" interactive learning environment in which ERT personnel can practice formal HazMat and first-responder tactics, procedures and protocols for a variety of real-world situations. Target users include members of HazMat response teams, such as firefighters, police officers, emergency medical services, and National Guard personnel. A major goal of the scenario development effort was to demonstrate that access to domain-specific content repositories pre-loaded with appropriate sets of reusable content components greatly facilitates the creation of a wide range of simulation-based, ADL training and education applications. As a result, each scenario was intended to showcase a different benefit of using component-based simulation technology. The first scenario recreated the Civil Support Team Trainer Sampling Mission of the original ECS ADL Prototype using a component-based approach. Replicating this scenario demonstrated the time savings that can be achieved when authors have access to libraries of pre-built components. In the second scenario, simulation components and assets from Scenario1 where reused to produce a new training simulation centered around a HazMat/WMD Recon Mission. As opposed to collecting samples of a HazMat/WMD agent in the Sampling Mission, the primary role of a HazMat Recon Mission was to survey the environment for hazards, locate the HazMat/WMD sources, identify the type of agent used and control and contain the agent release by isolating and/or disabling any dissemination devices used. The third scenario was a modification of Scenario2 and focused on Medical Recon as opposed to a HazMat Recon mission. In this scenario the team must triage civilians in the HotZone so they can be extricated to receive appropriate medical attention. This scenario was designed to demonstrate how a preexisting scenario can be modified and extended with additional behavior components to produce a completely different type of training experience.

SUMMARY OF RESULTS

soVoz has developed advanced behavioral animation technology [Refs. 7,8,9] that reduces the complexity of creating interactive 3D visualization and simulation content, encourages the reuse of such content in SCORM-conformant environments, and enables domain experts to directly participate in the authoring process. This technology was used as the simulation platform for the Prototype work.

PROTOTYPE OBJECTIVES

Project Objectives: The overall Prototype objective was to demonstrate that access to domain-specific content repositories pre-loaded with appropriate sets of reusable content components greatly facilitates the creation of a wide range of simulation-based, ADL training and education applications. The successful completion of the Prototype has:

- ***Advanced ADL instructional quality*** by making it much easier to create interactive training and education simulation content and scenarios. 3D game engine technology (i.e. NDL's Gamebryo) was used to provide state-of-the-art graphics and animation capabilities while high-level authoring tools enabled quick prototyping of interactive 3D content, allowing more time to test, assess and refine the effectiveness of the instructional design.
- ***Advanced ADL technologies and standards*** by proposing SCORM extensions supporting componentized simulation and testing using the SCORM sample runtime environment.
- ***Provided product demonstrations and transition*** by creating a library of simulation components and using them to demonstrate authoring of actual training content. Subject matter experts (SMEs) worked closely with soVoz to determine doctrinally correct training content and a baseline set of primary components. Initially, the application centered on specific tasks required to collect samples of hazardous materials (HazMat) in a Weapons of Mass Destruction/Emergency Response Team (WMD-ERT) training exercise. Once the prototype component library and infrastructure were complete, two

additional simulated scenarios were constructed to address other training requirements WMD-ERTs must train and prepare for. While the Prototype concentrated on WMD-ERT scenarios, it is important to note that the technology created is pertinent for a broad range of simulation-enriched Distributed Learning applications.

- ***Has applicability within and across services*** by demonstrating the effectiveness of advanced authoring tool and techniques for using shareable simulation content to create highly effective instruction and training environments.
- ***Provided a return on investment*** by (i) leveraging soVoz OSD-sponsored work and matching OSD SBIR Phase II+ funding in the areas of reusable simulation technology and tools, and (ii) by showing that access to domain-specific content repositories containing reusable simulation content can significantly reduce the time and cost of creating 3D training scenarios. The US Army Training and Doctrine Command (TRADOC) has established a goal to reduce the average time required for developing distributed training systems by 50% through the introduction of new tools, technologies, and procedures. The tools and methods developed through this project has made great strides in achieving this TRADOC objective.

Research Objectives: The Prototype met the following research objectives:

- ***Developed a prototype collection of 3D simulation components.*** This established a process and procedures for determining the minimal set of components required to develop domain-specific application content.
- ***Investigated extensions to the SCORM content model and runtime necessary to support a component-based approach to simulation***. There are a number of ways that SCORM-compliant simulation capabilities can be integrated with advanced distributed learning systems. This research objective resulted in a set of proposed SCORM extensions appropriate for supporting component-based simulation content.
- Identified and built the minimum set of necessary components required to construct a variety of emergency response team training scenarios. This allowed reusability of the component-based approach to be demonstrated. For example, components originally developed to model the operation of monitoring equipment could now be used to train a human technician on how to detect the presence of a hazardous material in a simulated WMD incident environment or be used by the artificial intelligence of a simulated HazMat team leader to determine whether decontamination procedures performed by other HazMat team members had been effective.
- ***Determined the "authorability" of interactive simulation content by instructional designers and Subject Matter Experts (SMEs).*** Given high-level authoring tools and a repository loaded with a rich set of simulation components, this research objective demonstrated potential improvements that could be realized by providing instructional designers and SMEs with a central role in the simulation scenario development process.

PROSCENA 3D COMPONENT SIMULATION TECHNOLOGY AND AUTHORING TOOLS

ProScena Technology is a high-level content authoring system and runtime for creating real-time interactive 3D simulations for training and education applications. The ProScena Technology solves three of the biggest problems associated with authoring interactive 3D content:

- It allows small development teams to benefit from state-of-the-art game engine technology (e.g. 3D graphics, behavioral animation, physics, popular gaming genres, etc.) without getting bogged down with low-level implementation details or infrastructure development,
- reduces the time, effort and skills required to produce engaging and effective training content by utilizing libraries of off-the-shelf content components, and
- manages the complexity of the 3D design process through an authoring environment that directly supports the needs of designers, artists, programmers and instructors. This helps parallelize the development effort, eliminates production bottlenecks, enables quick proto-typing of interactive content and allows the whole team to validate their ideas early and often.

System Architecture

ProScena Studio

ProScena Studio provides content developers with a set of high-level design tools, reusable content libraries and role-appropriate user interfaces for creating real-time interactive 3D simulation and training scenarios. In ProScena Studio, authors can express their plans either visually by pointing and clicking directly in the 3D scene; diagrammatically as a graph of event-driven behaviors; on a timeline or in a text-based scripting language such as Python. Since authors compose directly within the runtime environment, task sequences and decision logic can be readily tested and refined as soon as they are defined, making quick prototyping easy. ProScena Studio supports both 3DS Max and Maya plugins, allowing custom 3D art and animation assets to be easily imported and used. Programmers also can create new types of content components to support authoring of domain-specific application content. Once desired results have been achieved, everything from an individual movement to a full 3D scene can be annotated with descriptive (SCORM-conformant) metadata and saved back to a content repository for later reuse. ProScena Studio also allows instructors to fine-tune application content and customize it to their needs through a simplified version of the ProScena Studio user interface.

ProScena Player

The ProScena Player is the runtime that allows users to participate in the interactive 3D scenarios created with ProScena Studio. The Player is a platform-independent simulation engine responsible for coordinating the behaviors and actions of objects in a scene, rendering 3D graphics and audio, and supporting device I/O and network communication. It also provides a wide variety of asset management and lifecycle services intended to simplify the development process and maximize system performance. The ProScena Player can be hosted in other applications, such as web browsers, or run as a standalone application. The ProScena Player currently employs Numerical Design Limited's (NDL) Gamebryo game engine, enabling interactive 3D content authored in ProScena Studio to be deployed on a range of hardware and software platforms, from PC to game consoles.

ProScena SDK

At the heart of ProScena, is a powerful content model based on reusable components. Components act as wrappers for imported assets (e.g. 3D models, animation, sounds, procedural code, etc.) providing a simple and easy-to-use interface for content construction. The ProScena SDK allows developers to create basic simulation components in either Python or C++. An additional API is also provided to support third party expansion and customization. For example, the ProScena Player can be adapted to work with a variety of third party software, allowing interactive application content to be authored that is potentially reusable not only across hardware and software platforms, but also game engines. This capability allows existing game engines to immediately benefit from the ProScena suite of high-level authoring tools, enables previously developed application content to be repackaged and used in new ways and can help transition proprietary game or simulation engine technology to middleware solutions that are always up to date with state-of-the art features.

PROSCENA-BASED 3D SIMULATION CONTENT EXAMPLES

Three training scenarios were developed in this project to demonstrate the effectiveness of the soVoz component-based simulation approach. Each scenario showcases a different benefit of using component-based simulation technology. The scenarios were designed to support training of Emergency Response Teams (ERT) responding to hazardous materials incidents. The objective of the simulations was to provide a "hands-on" learning environment in which Emergency Response Team personnel can practice formal HazMat and first-responder procedures and protocols for a variety of real-world scenarios and situations. Students assume a role of a WMD-ERT team member responding to a terrorist attack. The training scenarios are intended to be relevant to members of HazMat response teams, such as firefighters, police officers, emergency medical services, and National Guard personnel. The focus of the scenarios was on recognizing and evaluating a hazardous materials incident, implementing basic control measures, refining decision-making skills, and protecting the public. Each scenario reuses the same collection of components, such as 3D models of the HazMat environment and response team personnel, HazMat equipment simulation and operation, contamination and exposure models, etc. to dramatically reduce the time and cost of authoring.

INTEGRATING SIMULATION-BASED APPLICATIONS WITH SCORM

This prototype was intended to serve as a research platform for exploring the use of interactive simulation in ADL applications. Through this research soVoz has demonstrated that simulations can be built in ways compatible with SCORM's Content Aggregation Model (CAM) and in accordance with its principles of "accessibility, adaptability, affordability, durability, interoperability, and reusability." Additional research was also conducted to determine the necessary set of extensions to the Run-Time Environment (RTE) to fully realize the potential of simulation in ADL applications.

Single and particularly multi-learner simulations place demands on SCORM that weren't accounted for in the original specification. It is readily apparent from soVoz's research that SCORM needs to be rethought if ADL applications are to exploit the full potential of simulation. soVoz's research for this project has explored in detail the issues associated with integrating single learner simulations into SCORM. Multi-learner simulations were also addressed but more work needs to be done. Two principal areas of research have been identified for further investigation.

- First, for SCORM to be relevant to simulation-based training, ways must be devised to allow the LMS to take an active role in determining the outcome of a simulation. Specifically, it must be able to sequence activities inside a simulation. Prior attempts to integrate simulation with SCORM have casually ignored one of SCORM's cardinal rules, which is that "content objects do not determine by themselves how to sequence/navigate through an aggregation representing a unit of instruction." Bringing the simulation under the control of the LMS implies that the interface points between content and LMS environments need to be reconsidered in light of simulation's unique characteristics. Simulations are composed of many complex rules and data dependencies. Most simulations today tend to be monolithic and inflexible because they mix scenario, mechanical, and assessment code into one package, failing to separate what should be orthogonal concerns. soVoz has been pioneering techniques that allow simulation developers to independently define and bind high-level training objectives, activities, and tasks to the simulation, thereby allowing greater reuse. When these techniques are applied to SCORM, the simulation would reconfigure itself to fulfill the learner's needs by consuming high-level training objectives, activities, and tasks encoded by the educator and fed to the simulation by the LMS. Future research should focus on extensions to SCORM that will allow the LMS to sequence simulations.
- Second, the set of services provided by the LMS needs to be enhanced and expanded to accommodate multi-learner simulations. SCORM does an adequate job of addressing single learner pedagogies; however, the standard is silent about multi-learner possibilities. Issues like group management, sequencing, and performance assessment all need to be reconciled against the existing standard. Additionally, new means of delivery and resource location will need to be devised to allow the RTE to control and coordinate a group's learning experience.

CONCLUSIONS

As part of a Phase II SBIR with the Office of the Secretary of Defense (OSD), soVoz has addressed many of the issues associated with creating and using reusable simulation content. This Prototype extended that work by designing and implementing a component collection for rapidly developing simulations for ADL applications. An emergency response team training scenario was authored to demonstrate the effectiveness of the approach. The research findings are summarized below.

CREATION OF INTERACTIVE SIMULATIONS FROM REUSABLE COMPONENTS

Creating interactive 3D simulations should be analogous to writing a script, assembling a cast of actors, giving the actors various roles to play, and directing their actions in a scene. Most of an author's time should be spent directing the actors in a scene, not "dressing the stage" by creating the sets, props, actors and individual actor movements. Current 3D creation techniques, unfortunately, require that this be done continually on a per project basis, using professional artists, animators and programmers to detail everything out before an instructional designer or training expert can even think about composing a 3D simulation scenario. The net result is to significantly increase development times and project costs while reducing the amount of time available for creating interactive simulations that maximize training effectiveness.

This prototype addressed this need by providing high-level authoring tools featuring point and click graphical user interfaces that allows interactive simulations to be:

- constructed using pre-existing building blocks, known as "actors" and "behaviors," accessed from libraries of reusable components,
- composed using authoring techniques such as sequencing, layering, blending and grouping of actors and behaviors,
- modified, adjusted and refined online by interactively changing behavior parameters, goals and triggering events,
- published to a content repository for later reuse and/or distribution by a Learning Management System

LEVERAGING COMMERCIAL GAME TECHNOLOGY AND AUTHORING TOOLS

This prototype made liberal use of third party software so as to not duplicate effort. soVoz Proscena Technology is built on top of numerous third party software products that provide basic services to the runtime. For graphics and device support we have employed NDL's Gamebryo, for audio Firelight Technologies' FMOD, and for networking MAK's VRLink. Each of these products provides a cutting edge implementation of their associated services. However, the soVoz runtime has been designed in such a way so as to allow each of these products to be replaced with a different implementation without affecting previously authored content. This gives content creators the peace of mind of knowing that their content will be preserved as the technology used to build it evolves.

The component-based approach researched in this Prototype could have benefited from a set of basic services that also subscribed to this philosophy. The graphics and networking toolkits employed, in particular, were not conceived as being used in a component-based environment. They assume that a C++ application would link statically to their libraries and not need to dynamically load and unload their services at runtime. We thus had to engineer these capabilities into these services, thus complicating the development effort. As the needs of this ADL community become more evident it is hoped that these vendors will build these features into their software designs, thus making it easier for third parties to build component-based systems on top of them.

RECOMMENDED SCORM EXTENSIONS

The Simulation as a Consumer of LMS Sequencing Information

In order to sequence a simulation the LMS will need to be able to observe the simulation state and issue sequencing commands in accordance with a defined learning strategy. To achieve this in the first phase of the project we developed a simulation agnostic means of communicating sequencing and tracking information between the LMS and a simulation.

Simulations maintain and generate an enormous amount of state information. In some cases, such as distributed deterministic simulation, the quantity is so great that it becomes impractical to send that information over the network to the RTE for sequencing. This data locality problem makes the server based sequencing in SCORM impractical for use with simulation. We propose a new distributed sequencing model whereby sequencing operations can be executed both remotely and locally where it is most practical. This implies that the role of the API Instance will expand to include sequencing and tracking operations. To observe data within the simulation a standard interface will be defined for communications between the simulation and the API Instance. The API Instance will need to have no specific information about the simulation it is controlling. It will issue a standard set of sequence commands to the simulation in response to state changes communicated from the simulation as a set of named events. We acknowledge that developers must be given maximum flexibility in building their simulations and would suggest an interaction model that makes as few demands on the simulation as possible.

Multi-learner Simulation in SCORM

As compared to single learner simulations, multi-learner simulations have vastly different learner management, sequencing, tracking, and assessment needs. To host multi-learner simulations the services of the LMS will have to be expanded.

- SCORM's Sequencing Specification needs to be extended to accommodate the non-linear and parallel nature of multi-learner simulation. Multi-learner sequencing needs to coordinate activities for a group or collection of groups and their members. Extensions to the CAM will have to be created to allow the description of multi-learner content packages. The Sequencer Service of the RTE will have to be enhanced to execute multi-learner activity trees.
- SCORM's Run-time Environment Data Model will also require an overhaul to properly track multi-learner simulations. The most obvious addition here is the quantity and type of tracking information that needs to be collected. Not only does the individual learner need to be tracked, but the activity of any group he or she is involved with also needs to be tracked. Additional data model elements will need to be added to the Run-time Environment Data Model to store this extra data. Both the LMS and simulation will need to update and query this information at run-time to operate at maximum potential. Additional support will need to be added to the API to allow cross learner access to this information.
- Some multi-learner simulations require teaming arrangements or other types of coordination in order to function. In addition to the LMS's traditional single learner profile management and course administration functions, next generation LMS's will

need to handle the orchestration of group learning. Match making will be an essential new service to facilitate multi-learner simulation. The match making services of the LMS can be used to connect similarly skilled learners, assemble a team of strangers for a group learning session, or assign autonomous agents as stand-ins for absent human counterparts.

- Finally, Multi-learner simulation will require frequent network communication to ensure that learners are properly synchronized. The mechanisms by which content is delivered and coordinated will need to be upgraded to support the bidirectional communication necessary for multi-learner simulation to occur.

This Final Report also includes a companion CD containing video demonstrations of the project results described above. The CD also contains a version of this Final Report document in the Adobe 5.0 PDF format.

SECTION II

SIMULATION CONTENT EXAMPLES

OVERVIEW

Three training scenarios were developed in this project to demonstrate the effectiveness of the soVoz component-based simulation approach. Each scenario showcases a different benefit of using component-based simulation technology. The scenarios were designed to support training of Emergency Response Teams (ERT) responding to hazardous materials incidents. The objective of the simulations was to provide a "hands-on" learning environment in which Emergency Response Team personnel could practice formal HazMat and first-responder procedures and protocols for a variety of real-world scenarios and situations. Students will assume a role of a WMD-ERT team member responding to a terrorist attack. The training scenarios are intended to be relevant to members of HazMat response teams, such as firefighters, police officers, emergency medical services, and National Guard personnel. The focus of the scenarios is on recognizing and evaluating a hazardous materials incident, implementing basic control measures, refining decision-making skills, and protecting the public. Each scenario reuses the same collection of components, such as 3D models of the HazMat environment and response team personnel, HazMat equipment simulation and operation, contamination and exposure models, etc. to dramatically reduce the time and cost of authoring.

ECS Civil Support Team Trainer (CSTT)

The project leveraged previous simulation development efforts by Engineering & Computer Simulations, Inc. (ECS) in the area of Civil Support Team Training (CSTT). Using a *"Structured Training"* approach, ECS's CSTT system interactively navigates the student through a series of training tasks while communicating the student's actions to the 3D simulated environment. Structured training provides a deliberate focus on training objectives by immersing participants in a realistic scenario, with cues and conditions set up to support a planned sequence of task performance.

The prototype CST simulation developed by ECS was based on the CST Training and Evaluations Outline (T&EO) entitled Conduct Chemical/Biological Survey Operations (***ARTEP 3-627-35-MTP)***. This T&EO can be further broken down into eighteen (18) individual Task Steps and Performance Measures as outlined in the "***Certification of WMD Civil Support Teams (CST) O&O Concept and ARTEP"*** The primary actions for the Survey T&EO Task are separated into three areas:

1. Actions before the Survey and Entry.
2. Actions During the Survey
3. Actions after the Survey

Actions during the Survey were the main focus of the ECS Prototype. Specifically, HazMat/WMD agent sample collection.

In this project, pre-existing 3D models of urban terrain and emergency response team personnel, animations and descriptions of operational procedures relevant to the T&EO sampling task were packaged as components and incorporated into a simulation repository as Actors and Behaviors. They were used as the starting point for developing three training scenarios involving HazMat equipment use and procedures, WMD incident awareness, operations and management tasks. To quantify the effectiveness of the component-based approach in terms of return on investment, ECS's Civil Support Team Trainer sampling task simulation provided a baseline for comparison. For example, assuming that 3D models already exist for the scenario environment and each team member, the amount of time required to create the application content for a particular training scenario using a traditional programming-based approach (i.e. ECS) vs. the proposed component-based simulation approach will be analyzed. Throughout the development effort careful evaluations will be made of how simulation components simplify the authoring process, improve simulation functionality and enhance training effectiveness by supporting performance-based instruction.

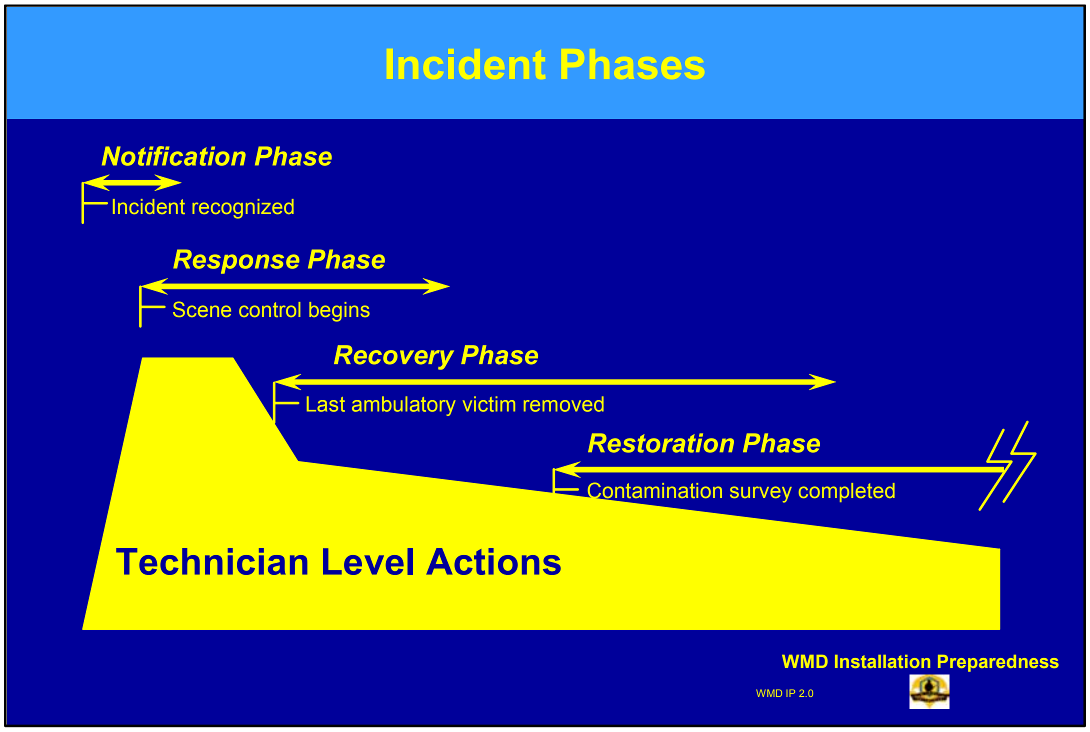
Emergency Response Team Training

Overview

In the event a terrorist employs a chemical, biological, or radiological device, emergency response teams consisting of police, fire fighter and emergency medical personnel are going to be on their own for the initial critical phases of the response. Because of this, knowledge of proper procedures and responder actions can significantly limit the extent of the incident. Therefore, the simulation scenarios developed in this project were intended to be used for training not only Civil Support Teams but also Emergency Response Teams at the HazMat Technician level.

HazMat/WMD Incident Phases

Responses to a HazMat/WMD incident can generally be broken down into four phases: notification, response, recovery, and restoration. Of the actions which occur during these phases, responders at the HazMat Technician Level are typically involved with those actions which are offensive in nature. The figure below depicts the level of work intensity of Technician Level responders in response to a HazMat/WMD terrorist incident: high intensity during the response phase then tapering off as the incident is resolved.



1. **<u>Notification Phase</u>.** The notification phase begins with recognition that an incident has occurred, and continues until the first emergency vehicle arrives and response personnel begin site management. The notification may be made by anyone. It might not come from a responder trained to the Awareness Level, but rather from an individual who notices the development of a mass casualty situation.

   - This phase should optimally conclude within 15-30 minutes. The first priority is, as always, responder safety. During this time, information must be gathered, particularly the location of the incident and the wind direction, because it is critical that all responders approach incidents involving WMD agents from the upwind and upgrade direction. Responders should also begin site management, to include site security, and begin to restrict movement into the downwind hazard zone.

   - Only ambulatory casualties are evacuated during this phase. Emergency decontamination of mass casualties is initiated by corralling them, setting up the decontamination corridor, and having the victims remove their clothing so responders can flush them with large amounts of water. The victims are then provided covering and first aid.

2. **<u>Response Phase.</u>** The response phase starts with the beginning of scene control. This phase will typically last several hours and focuses on the saving of lives. It includes actions such as rescue, emergency decontamination of mass casualties, agent identification, evidence collection, searches for secondary devices and perpetrators, and the emergency decontamination or neutralization of large concentrations of agent.

3. **<u>Recovery Phase</u>.** The recovery phase begins after the last living casualty has been evacuated from the hazard area. During this time, the focus is on reestablishing the essential services

which may have been interrupted by the incident. It will include actions such as establishment of an equipment decontamination corridor and decontamination of essential equipment. It is during this phase that state and federal responders may arrive to provide assistance, though some may have already arrived. At this time, fatalities are moved to the Hot Line pending instructions for the disposal of the remains.

4. **Restoration Phase.** The last phase, the restoration phase, begins upon completion of the survey for contamination and continues until all contamination has been eliminated. The focus will be on restoration of the site to its original state, with emphasis on site safety.

Typical Response Phase Tasks

Since the Response Phase has the highest risk to responders and is the most complex and requires tactical decision making, it was used as the context for the scenario designs. During the Response Phase emergency response HazMat teams typically perform the following tasks and actions:

- **Don appropriate Personal Protective Equipment (PPE) for Hot Zone and Warm Zone activities.** Don PPE appropriate for the hazard involved and the task to be performed. Generally, the initial response in the Hot Zone will be in Level A.
- **Conduct a Hazard and Risk Assessment**. Determine the type of HazMat substance or WMD agent released in the environment. A key part of this is an initial search of the HotZone (i.e. Recon) to detect and assess the extent of the hazard. A key part of this assessment might be the decision to evacuate or shelter in place.
- **Establish, supervise, and operate emergency and technical decontamination.** Set up, operate, and/or supervise the operation of the emergency and technical personnel decontamination corridors
- **Rescue victims in Hot Zone.** This is often a difficult task for Hazmat teams in Level A protection. If emergency medical responders are trained and equipped to perform this mission, non-medical HAZMAT personnel may be better able to perform the more technical aspects of the mission.
    - Ambulatory victims in the Hot Zone should be immediately directed to the Warm Zone for decontamination.
    - Non-ambulatory victims should be tagged for triage according to the start system
    - Victims should be given appropriate medical attention and extricated for decontamination
- **Ensure medical support for HAZMAT operations in the Warm and Cold Zones.** Ensure casualties and responders are provided with appropriate medical care. Assist by providing first aid, within one's capabilities and training, after decontamination and evacuation.
- **Be alert for secondary devices, multiple incidents, and perpetrators.** During the initial site Recon, be alert for secondary devices and perpetrators. Perpetrators often place devices above the ground and near hydrants, trash cans, dumpsters, mail boxes, etc. to maximize destruction effectiveness.

- **Attempt to identify the agent.** Many actions hinge on the fast and accurate identification of the agent or agent type employed. Treatment of casualties can be more specific, PPE levels can be downgraded, and evacuation or shelter-in-place decisions can be made more cogently after the threat agent is identified.
- **Collect agent samples.** Sample management ensures that a sample is verifiable by having proper collection, labeling, and documentation data, as well as packaging and transport to preserve the sample and move it to a laboratory equipped to conduct an analysis. Samples of the agent and of anything which is confirmed or suspected to be contaminated should be collected. Both the sampling of the agent and the identification and evacuation of the dissemination device should ideally occur before any remediation efforts. Balance sampling with neutralization, realizing that if the hazard is not neutralized quickly, the downwind hazard area may continue to grow.
- **Assist evidence collection.** Assist with the efforts to collect other evidence from the site. The need to collect, safeguard, and accurately document evidence, both to identify and to prosecute, is a vital part of any response. Preserve evidence as much as possible, remembering that when neutralizing the source of the hazard, evidence which could be used later in apprehending and prosecuting the perpetrators might be destroyed.
- **Perform mitigation actions** Locate, isolate, contain, and neutralize large areas of contamination, employing mitigation techniques. As soon as evidence collection is complete (or before, if the hazard is determined to be of such magnitude that evidence collection becomes a secondary issue), large areas of contamination which contribute to the generation of a downwind vapor hazard should be isolated and neutralized. The sooner the volatilization process can be stopped, the sooner the downwind hazard can be mitigated.

HazMat Simulation Scenario Tasks

The three scenarios developed in this project involve the following tasks normally performed by HazMat emergency response teams during the Response Phase of incident management:

- Conduct a Hazard and Risk Assessment
- Rescue victims in Hot Zone
- Be alert for secondary devices, multiple incidents, and perpetrators
- Attempt to identify the agent
- Collect agent samples

Simulation-Based Training

Training Audience

Since the HazMat Scenarios developed in this project require entry into a Hot Zone, it is assumed that only those individuals who have been previously trained to operate in or near the Hot Zone have the necessary knowledge to perform the tasks required in the simulation scenarios. Therefore the intended audience for the HazMat scenarios is assumed to be emergency response personnel already at the operations- or technician-level of training. It is assumed that HazMat Scenario1 can be done by HazMat technicians. HazMat Scenario 2 (Recon) and HazMat Scenario 3 can be done by firefighters and/or HazMat operations personnel, including EMTs and HazMat technicians. There are no present tasks in the scenarios that require law enforcement participation, although future scenarios involving evidence collection, search for terrorists, etc., could also be investigated.

Instructional Design

The overall approach to the instructional design of the simulated scenarios was to decompose the training horizontally into task modules, and design each task module vertically with various levels of instruction and performance assessment matched to a participant's knowledge and abilities (i.e. skill levels). The task module approach allows participants to focus their efforts on a specific set of tasks, such as collecting a HazMat sample, using detection equipment, identifying and extricating casualties from the Hot Zone, etc. This mirrors the "Lanes" type lessons and approach given at the Center for Domestic Preparedness. Modules for HazMat detection and sample collection support Scenario 1. A search module supports Scenario 2 (Recon), while a rescue module supports HazMat Scenario 3 (Medical Recon).

The vertical separation into skill levels supports different methods of presentation and user interaction; initially demonstration-based at a slow pace, then self-driven with increasing levels of student interaction and independent decision-making. Three skill level categories can be used to support such simulation-based instruction: **Demonstration-based Instruction**, **Performance-based Instruction** and **Exercise**. Although in this project the simulation-based instruction consist mostly at the Demonstration and Exercise Skill levels, descriptions of the characteristics of all three types are provided in this final report for completeness.

<u>Skill Level 1: Demonstration-based Instruction</u>

In the Demonstration-based Instruction phase a virtual instructor (VI) describes the task to be performed and teaches basic tactics and techniques, presuming the student participant has been previously familiarized with the operation of any equipment used. The method of instruction is primarily of the form of a lecture/demonstration. The virtual responder (VR) representing the student in the simulation performs the actions as instructed by the VI and has limited ability to do anything else. That is, the VRs focus of attention is primarily on the task at hand. The goal of the Demonstration-based Instruction phase is primarily on teaching the student tactics, procedures and techniques rather than actual manipulation skills or the physical operation of specific devices or equipment. This type of training is always better performed using a hands-on approach with the actual device, kit or protective equipment and non-lethal HazMat agent simulants. Users need to have performed all sub-tasks in a module correctly in order to complete the lesson. The Demonstration-based Instruction phase is characterized in terms of the following:

- **Flow of training**. The virtual instructor teaches or demonstrates the instruction point or sub-task, then directs the operator to perform a related action. Each sub-task in a module is done to completion, one step at a time at the virtual instructor's pace. If desired, a sub-task can be repeated using minor variations. For example, collecting a liquid sample with the syringe is a different subtask from collecting a liquid sample with the swatch and forceps.

- **Visuals**. The background environment can be simple, such as a plain space (e.g., field, parking lot) with an obvious object of concern, such as a broken HazMat container. No complications such as victims, explosives, or other threats are needed. Depending upon the task the virtual instructor may or may not need to be visible. The virtual responders appear in Level A personal protective equipment (PPE) to restrict their visibility and dexterity to that used in the Exercises. Windows that display task descriptions, control panels and checklists are always available. When appropriate, a graphical icon can appear in the environment to direct the user's attention or show an appropriate location to place or position an object. For example, a dotted outline indicating where to place the tarp when preparing to collect a sample. This is comparable to a real instructor placing the tarp, then removing it to allow the student to place it correctly. Also when appropriate, a pop-up window containing a checklist can appear, and be checked off as each step in the task module is completed.
- **User Interaction**. Users have limited control over their virtual responder. The virtual instructor can direct the virtual responder's point of view to look at specific objects of interest in the 3D world, highlight buttons and icons on the user control panel and filter user inputs so they can only perform the desired actions. Other (incorrect) input responses can be either locked out or ignored. Once the user performs the correct action they get positive feedback from the virtual instructor and observe its effect on the world.
- **Performance Assessment**. The virtual instructor provides feedback to help the user when they are having difficulty performing the subtask. For example, a mistake can trigger a "beep" or prompt a more detailed voiceover that repeats the instructions for that sub-task or procedure. In cases where the student can not perform the required sub-task after a number of tries, the virtual instructor can demonstrates the proper procedure again.
- **Supervisor Control**. The supervisor will have the ability to vary the basic setup of a training scenario to ensure mastery of the task at hand as opposed to learning how to manipulate the training simulation. A supervisor-override mode may also be provided to unlock any repetitive cycles that users may fall into before they become frustrated with using the training simulation.

Skill Level 2: Performance-based Instruction

Performance-based Instruction allows the responder to perform the basic tactics and techniques previously learned in the Demonstration-based Instruction phase. However, the user has complete control of the virtual responder's actions and the virtual instructor guides the user through the performance of the task when they are having difficulty (i.e. intelligent tutor), much like a real instructor standing over their shoulder talking them through the process. Users need to have performed all sub-tasks in a module correctly in order to complete the lesson

- **Flow of training**. Each sub-task is done to completion, step-by step at the user's pace. If desired, a sub-task or step can be repeated when a user makes a mistake. For example, using a chemical detector in a manner that would not get the expected reading can be allowed, but the operator would be prompted to do it again the right way. Alternately, the operator can be warned of impending danger if the virtual responder is about to do something dangerous.

- **Visuals**. The background environment can be either the simple environment from the Demonstration-based Instruction phase or the full 3D terrain of the simulation scenario environment used in the Exercise training phase. If the full scenario environment is used, complications resulting from victims, explosives, or other threats can be present to show realism, but these should be minimized so they do not interfere with the task being trained. Depending upon the task the virtual instructor may or may not need to be visible. As before, the virtual responders appear in Level A personal protective equipment (PPE) to restrict their visibility and dexterity. Control panels showing command selections are always displayed. Checklists can be either always displayed (as in Demonstration-based Instruction) or made available through a pop-up window upon request. When appropriate, a graphical icon can appear in the environment to direct the user's attention or show an appropriate location to place or position an object.
- **User Interaction**. Users have complete control over their virtual responder. When in the process of providing performance-based instruction, the virtual instructor can direct the virtual responder's point of view to look at specific objects of interest in the 3D world, and/or highlight buttons and icons on the user control panel. Once the user performs the correct action they get positive feedback from the virtual instructor and observe its effect on the world.
- **Performance Assessment**. The user can request help, either in the form of a task description, checklist, virtual instructor "guidance" (e.g. suggestions about what they are doing wrong) or by having the instructor demonstrate a sub-task they find particularly difficult. The virtual instructor can also alert the user to the presence of potentially dangerous conditions or situations, notify them when they are about to fail a critical task, or cause injury to the virtual responder (or their virtual buddy). For example, if the user is about to cause the virtual responder to step into a spill, the instructor voiceover can say, "Stop, you are about to step into the hazard!". For non-critical sub-tasks, the voiceover can flag incorrect responses and suggest appropriate actions or a better way of doing a something. The virtual instructor can also allow completion of an incorrectly performed sub-task and then direct the operator to repeat it correctly.
- **Supervisor Control**. Same as Demonstration-based Instruction.

Skill Level 3: Exercise

This phase can use a coach to help the operator in the performance of the task when needed, much like a real instructor on the other end of a radio, watching from a distance.

- **Flow of training**. Similar to Performance-based Instruction but users must perform tasks subject to a time constraint..
- **Visuals**. The background environment is the full 3D terrain of the simulation scenario. Complications resulting from victims, explosives, or other threats are also present. The virtual instructor does not appear until the after action review, but may provide some voiceovers during the training if a critical task or mission has failed. This is not the same

voice as the radio commander, who may request status, provide information or give the HazMat team/team members directions to perform specific tasks or actions. As before, the virtual responders appear in Level A personal protective equipment (PPE) to restrict their visibility and dexterity. Control panels showing command selections are always displayed. Pop-up window Checklists are only available upon request.

- **User Interaction**. Users have complete control over their virtual responder. There is no performance-based feedback and instruction from the virtual instructor.
- **Performance Assessment**. Incorrect responses are flagged for discussion in the After Action Review. A virtual instructor voiceover will alert the user in cases where improper performance of a critical task has lead to mission failure. Upon task/mission failure or completion, the virtual instructor will critique the user's performance in an After Action Review, covering each step that was done correctly and incorrectly. For incorrect or missing actions, the virtual instructor will describe the proper warning signs, decision points, and/or operating procedures. The user can also be graded in the performance of the task, much like a certification exam.
- **Supervisor Control**. Same as Demonstration-Based Instruction.

Adjustable Scenario Parameters

Each scenario will have a set of basic parameters that a supervisor can easily adjust to change the configuration and details of the training exercise. Parameters include: lighting, terrain features, buildings, population, and the initial conditions of the HazMat agent release. HazMat agent parameters can include the type (e.g. which chemical or bio material to use), dissemination method (e.g., explosive, pressurized spray, spill, etc.), release point(s), quantity of material to be release, state of matter (e.g., powder aerosol, viscous liquid, volatile liquid, etc.), extent of contamination with respect to the release point, and present status of the HazMat agent when encountered in the Hot Zone by the HazMat team. Other parameters include the location and condition of victims, obstacles, location and detonation of secondary devices, etc.

SCENARIO1 – HAZMAT/WMD SAMPLING MISSION

Scenario Description

Overview

The first scenario will replicate the tasks provided in the Civil Support Team Trainer developed in ECS's previous ADL prototype effort at Skill Levels 1 and 3 described in Section 2.2. In Scenario1, a HazMat Survey Team consisting of a Survey Team Leader, Sampler and Assistant Sampler, is directed to enter the Hot Zone and obtain samples of suspected HazMat/WMD chemical agents. A HazMat Section Officer also communicates with the team when they are in the Hot Zone and directs their operations when and if appropriate. To succeed, the team members must collaborate with one another to complete their mission using team monitoring and testing equipment as well as employing proper sampling procedures. Replicating this scenario will demonstrate a reduction in development time by using a component-based approach. This scenario is intended for a multi-player mode of operation.

References

- C/B Incident Response Sampling Kit operating instructions.
- APD-2000 Users' Manual.
- ARTEP 3-207-10-Drill 03/14/2002 Drills For The Nuclear, Biological, Chemical (NBC) Reconnaissance Platoon
- ARTEP 3-627-35-MTP 06/22/2001 Mission Training Plan For The Weapons Of Mass Destruction (WMD) - Civil-Support Team (CST)
- Army FM 3-11.19 07/30/2004 Multiservice Tactics, Techniques, And Procedures For Nuclear, Biological, And Chemical Reconnaissance
- Army FM 3-11.22 Appendix L, 06/06/2003 Weapons Of Mass Destruction Civil Support Team Tactics, Techniques, And Procedures

Prerequisite Instruction/Knowledge

Previous completion of a minimum 40 hour hazmat technician level training course, proper understanding and operation of the following equipment and devices: Level "A" entry suit, 1-hr. SCBA, APD2000, ICAM, Multi-RAE, M256 Kit, pH & M9 paper, photo-ionization detector (PID), Hazmat ID, and radiation instrument.

Scenario Design Goals and Training Objectives

Design Goals

The intent of the simulation Scenario1 is to provide HazMat responders with a working knowledge of the procedures and equipment used for the detection and collection of samples from a chemical source of a HazMat/WMD Weapons event.

Training Objectives

- Locate a chemical agent for sampling
- Demonstrate proper operation of HazMat/agent detection and sampling equipment
- Demonstrate proper sample collection procedures and protocols
- Implement concept of Clean man/Dirty man whereby the Sampler is the only member of the team to have contact with the suspected hazardous material.
- Maintain situational awareness during performance of task

Individual and Team Task Descriptions

Mission Objective

The mission of the HazMat Survey Team in Scenario1 is primarily to collect samples from areas known or suspected of being contaminated with toxic chemical agents. These samples are collected for the purpose of analyzing and determining the extent of the contamination. This information could ultimately inform decisions on mitigation, evacuation and medical treatment as well as other goals and objectives of the Incident Commander. The HazMat/agent sample collection task consists of a series of interactions between the survey sampler (denoted SS), assistant sampler (denoted AS) and team leader (denoted TL) HazMat team members.

The following is the task sequence that must be followed by various individuals on the HazMat team:

HazMat Sampling Mission Tasks (TL = Team Leader, SS = Sampler, AS= Assistant Sampler)

	SubTasks	***Steps***	***SubSteps***
1	*Enter HotZone*	*SS: Pick up hazardous waste container and tarp* *AS: Pickup tools and equipment* *SS and AS: Move to the potential site of HazMat/agent* *SS: Put down the waste container.* *SS: Place the tarp near the hazard site* *AS: Place tools, equipment and waste container on tarp after it is placed in position.*	
		Decon hands	*1. AS: selects bleach decon rag and hands to SS* *2. SS: Uses bleach decon rag to wipe hands* *3. SS: Discards bleach decon rag into the hazardous waste container.*
2	*Locate a chemical agent for sampling*	*Visually scan the area to locate suspected liquids.* *Test a potential liquid HazMat source using M-8 paper.* *Test a potential vapor source using APD-2000.*	
3	*Prepare to collect a HazMat/agent sample*	*1. SS: Places ruler next to the area to be sampled.* *2. SS: Determines type of sample to be taken* *3. AS: pickups up sample collection jar from drop-cloth and hands it to Team Leader.* *4. TL: uses radio to start Chain of Custody. Reads Sample Identification number.* *5. TL: gives sample collection jar back to AS.*	

	SubTasks	Steps	SubSteps
		6. *TL: takes two pictures of target area to be sampled. Photograph must include sample area, sample collection jar and ruler.*	
4	*Collect a liquid chemical HazMat/agent sample.*	1. *Collect HazMat /agent samples*	1. *AS: Places one sample jar and tamper resistant seal on tarp near SS and hands sample jar to SS.* 2. *AS: hands SS either swab, wipe or syringe as requested* 3. *SS: collects sample by positioning swab, wipe or syringe near sample collection device* 4. *SS: places swab or wipe into the hazardous waste container.* 5. *AS: seals jar with a security seal*
		2. *Place HazMat/agent samples in Overpack Bag*	1. *AS: hands SS bleach rag* 2. *SS decontaminates the sample jar with bleach rag* 3. *SS: places bleach rag into the hazardous waste container.* 4. *AS: gives SS a seal for sample jar* 5. *SS: seals sample jar.* 6. *AS: picks up overpacking material from dropcloth* 7. *SS: hands sample jar to AS.* 8. *AS: places sample jar into the overpacking material* 9. *AS: Place overpack material into overpack bag.*
		3. *Complete sample collection*	1. *Repeat Steps 1 and 2 above until a total of three samples have been collected* 2. *AS: overpack camera and place into carry bag.* 3. *AS: assist SS in closing sample overpack bag.* 4. *SS: Clean gloves using decon bleach rag* 5. *AS: Pick up equipment.*
5	*Exit the Hot Zone.*	1. *Move to Warm Zone Decon area.*	

User Interaction Models

The HazMat team must enter the Hot Zone, search for and locate sites of the suspect chemical liquid using chemical agent detection equipment, and collect a sample of the liquid using the C/B Incident Response Sampling Kit. Users assume the roles of the team leader (TL), sampler (SS) or assistant sampler (AS) and must follow the instructions given by the virtual instructor (VI). It is assumed that the HazMat Section Officer has given the team all the information available, including initial descriptions of the incident, possible locations of hazardous substances and the number of known casualties. Students participate in the simulation as if they were the person performing their assigned duties.

Skill Level 1: Demonstration-based Instruction

Students have limited control over the simulation. For example, the simulation may contain the presence of other responders and objects within the student's immediate view. However, these do not respond to student actions. Initially, students are able to observe the correct response being performed without being able to influence the action. As the training progresses, students use the mouse and/or keyboard to perform actions as directed, following the basic simulation operating procedures. Incorrect responses will not be allowed during this phase of instruction. The student may have to command the start of an action but will not have to demonstrate physical dexterity to complete the performance step. The simulation will play the role of the virtual instructor, which will be present as visual or voiceover throughout the training.

User Interface

The user commands the movements and actions of an avatar (i.e. virtual responder) that represents their presence in the 3D world. Users can select between a first person (looking through a Level "A" entry suit) point of view or a digital camera point of view when controlling their virtual responder.

Instruction given to Students/Users

As per the instructional design described in Section 2.2.2, users will learn proper operation of the Chemical/Biological (C/B) Incident Response Sampling Kit and procedures for collecting HatMat/agent samples with guiding feedback from a virtual instructor. That is, users can not proceed to the next step in the sampling task sequence of Section 3.1.4 above until they perform each step correctly.

Scenario Simulation

This scenario assumes the Hazmat Team is already outfitted in Level A Personal Protective Equipment (PPE). Responders start outside a marked area. The area includes a simulated spill of a liquid chemical. The Virtual Instructor (VI) appears in front of the Virtual Responder (VR) view. The Other Virtual Responders (OVR) in the team stand nearby. Assigned equipment is on the ground nearby. Students begin when directed to proceed. The actual sequence of actions and VI dialog will be drawn from the existing ECS prototype.

SubTask1. Enter Hot Zone.

Steps	*Instruction Dialog/Actions*
SS: Pick up hazardous waste container and tarp *AS: Pickup tools and equipment*	[VI voices each command and points toward each item or location of interest. VR briefly looks in that direction, then back to VI until dialog is complete.]
	[VI recommends which items to be picked up by VR. Other VR picks up remaining items.]
	[VR looks at each item in turn.]
	[VR looks at tool box, window appears, highlights button, status changed to show item picked up, repeat for bucket.]
SS and AS: Move to the potential site of HazMat/agent	[VR turns to look at spill, moves in direction of spill site, stops near spill.]
SS: Put down the waste container.	[VI voice over, VR does as directed, student observes placement of trash bucket and tarp.]
SS: Place the tarp near the hazard site	[VI voice over, VR does as directed, student observes placement of trash bucket and tarp.]
AS: Place tools, equipment and waste container on tarp after it is placed in position.	[VR puts down tools on tarp, centered in VR view. Window appears, button highlighted, status changed to show items carried minus toos.]
	[VI directs student to perform this function and waits until completed. VR responds to student control; windows appear as commanded. If student correctly positions bucket on tarp, VI adds, "Very good." If student attempts to place bucket outside of tarp area, VI voice suggests, "Try again." If Student attempts to move VR to another location or select another command, VR will not respond and VI voice suggests, "Try again." Continue until VR has placed bucket correctly.]
Decon hands	[VI directs students to decontaminate hands using bleach rag

SubTask2. Locate a liquid chemical agent for sampling.

Steps	*Instruction Dialog/Actions*
Visually scan the area to locate suspected liquids.	[This is initially all demo, VRs controlled by VI. Voiceovers as appropriate]
	[VI asks student to try]
Test a potential liquid HazMat source using M-8 paper.	[This is initially all demo, VR and other VR controlled by VI. Voiceovers as appropriate]
	[VI asks student to try]
Test a potential vapor source using APD-2000.	[This is initially all demo, VR and OVR controlled by VI. Voiceovers as appropriate]

	[VI asks student to try]

SubTask3. Prepare to collect a HazMat/agent sample

Steps	***Instruction Dialog/Actions***
Places ruler next to the area to be sampled.	[VI directs student where to place ruler]
SS: Determines type of sample to be taken	[VI informs student re: type of sample to be taken.]
AS: pickups up sample collection jar from drop-cloth and hands it to Team Leader.	[VI indicates location of sample jar on drop cloth and directs student to pick it up and hand it to TL]
TL: uses radio to start Chain of Custody. Reads Sample Identification number.	[VI tells TL info that needs to be communicated to HazMat Section Officer. Asks TL to radio info then give jar back to SS.]
TL: gives sample collection jar back to AS.	
TL: takes two pictures of target area to be sampled.	VI directs TL to take two pictures of sample area, making sure photograph contains sample area, sample collection jar and ruler.]

SubTask4. Collect a liquid chemical HazMat/agent sample.

Steps	***Instruction Dialog/Actions***
Collect HazMat /agent samples	[This is all demo, VRs controlled by VI. Voiceovers as appropriate]
	[VI asks student to try]
Place HazMat/agent samples in Overpack Bag	[This is all demo, VRs controlled by VI. Voiceovers as appropriate]
	[VI asks student to try]
Complete sample collection	[This is all demo, VRs controlled by VI. Voiceovers as appropriate]
	[VI asks student to try]

SubTask5. Exit Hot Zone.

Steps	***Instruction Dialog/Actions***
Move to Warm Zone Decon area.	[VI directs students to move to the Warm Zone for Decon]

<u>On-air time</u>

Not applicable. The user's virtual responder on-air time will not be monitored during this phase of training. Students are not timed in the performance of tasks.

Accidents and Dangerous Situations

Not applicable. No simulated accidents or injuries occur as the result of chance, student action, or inaction.

Duration

Not applicable.

Skill Level 2: Performance-based Instruction

Not applicable. Beyond the scope of this prototype effort.

Skill Level 3: Exercise

Users perform the sequence of tasks without any intervention from the virtual instructor as described in the instruction design of Section 2.2.2. HazMat/WMD agent samples are placed in different locations in the environment and the team needs to find these, locations, identify the type of HazMat/agent and collect a sample subject to time/air constraints.

On-air time

The user's virtual responder will have 60 minutes of simulated air time to complete the sampling mission during this phase of training.

Accidents and Dangerous Situations

Not applicable. No simulated accidents or injuries will occur as the result of chance, student action, or inaction.

Duration

60 Minutes

Adjustable Scenario Parameters

Each scenario will have a set of basic parameters that a supervisor/trainer can easily adjust to change the configuration and environmental details of the training exercise.

Scenario Configuration

Environmental parameters for Scenario1 include the initial conditions of the HazMat agent release. HazMat agent parameters include the type (e.g. which chemical material to use), dissemination method (e.g.pressurized spray, spill, etc.), release point(s), quantity of material to be released, extent of contamination with respect to the release point, and present state of the HazMat agent when encountered in the Hot Zone by the HazMat team.

Accidents and Dangerous Situations

None.

Performance Assessment

The following table outlines the evaluation criteria that the application uses to determine if each task in the scenario has been completed correctly. For each incorrectly performed task points are deducted from the learner's total score.

	Task Steps	***Performance Measures***
1	***Team maintains situational awareness***	• *Maintained the buddy system and checks.* • *Remained within line of sight of each other.* • *Conducted continuous hazard monitoring.*
2	***Team conducts mission tasks according to plan***	• *Detected and identified HazMat/WMD agents that were present in environment subject to mission requirements.* • *Samples collected according to established practices and procedures.* • *Established and maintained military chain of custody (positive control)*
3	***Team maintains communications***	• *Provided situation reports (SITREPs) to HazMat section officer when appropriate.*

Skill-Level 1. Students must complete all subtask steps before they can move on to the next subtask. Students may repeat any step in a subtask without direction from the virtual instructor. When the student has completed all subtasks in sequence without error, the student can advance to Skill Level 3, the exercise phase of instruction.

Skill-Level 2. Not applicable.

Skill-Level 3. The virtual instructor will summarize the teams performance in an after action review after all responders have left the Hot Zone.

SCENARIO2 – HAZMAT/WMD RECON MISSION

Scenario Description

Overview

This Scenario reuses assets from Scenario1 and produced an entirely different trainer centered on a HazMat/WMD Recon Mission. The primary role of a HazMat Recon Mission is to define the Hot Zone perimeter, use instruments to determine oxygen levels and the presence of a flammable and/or toxic environment and to specifically detect and identify a HazMat/WMD agent and/or dissemination device. The HazMat Recon entry team normally consists of two members; one being the team leader. The two-member entry team enters the Hot Zone and surveys the environment for hazards, locates the source of the WMD agent release and controls, contains and possibly transports an agent sample/or device back out to the Warm Zone. For each two-person Recon Team that enters the Hot Zone, there is also a two-person Hazmat Rescue team on standby and on-air, just outside the Hot Zone who can instantly respond to any Recon team member who has a failure of their personal protective equipment, becomes injured or ill, or is struck by a secondary device. When commanded, this Rescue team locates the Recon entry team members in the Hot Zone and escorts the fallen team member (or packages them on a stretcher) and evacuates them to the Warm Zone Decon area. A 5th member of the Hazmat Team serves as the Hazmat Sector Officer who communicates with the HazMat Recon team in the Hot Zone and directs their operations if and when appropriate. To succeed, the team members must collaborate with one another to complete their mission using monitoring and testing equipment as well as employing proper search, survey and containment procedures. This scenario will support single-player and multi-player modes of operation.

References

- "WMD Hazmat Technician Course", Installation Preparedness (IP) Program U.S. Marine Corps, 2003
- "EPA Hazmat Technician Course", United States Environmental Protection Agency, 2002
- FM 3-11.22 Appendix L, 06/06/2003 Weapons Of Mass Destruction Civil Support Team Tactics, Techniques, And Procedures

Pre-Requisite Instruction/Knowledge

Previous completion of minimum 40 hour hazmat technician and paramedic course with additional training, proper understanding and operation of Level "A" entry suit, 1-hr. SCBA, APD2000, Sabre 4000, M256 Kit, Multirae, Bioseeq, Radmeter, Dosimeter, Immuno Assay Tickets, Draeger Tubes, photo-ionization detector (PID), Hazmat ID Gas ID, and TI Comannder. Completion of the virtual scenario instruction, using all items, is required

Scenario Design Goals and Training Objectives

Design Goals

The intent of simulation Scenario2 is to provide HazMat responders with a working knowledge of proper operating procedures and agent detection/identification equipment when searching and surveying a location suspected of having been contaminated by an unknown HazMat/WMD subject to defined time constraints.

Training Objectives

- Effectively execute all critical mission tasks within the time restraint of the Level "A" self-contained breathing apparatus (SCBA) 1 hour air bottle (mandatory)
- Operate the thermal-imaging camera to locate suspicious packages, dissemination devices, and chemical spills within the determined "Hot Zone" of the attack scene (mandatory)
- Conduct a recon for contaminants within the determined "Hot Zone" of the attack scene (mandatory)
- Operate the proper chemical, biological, radiological, or explosive detection & ID instrumentation for the mission requirements (mandatory)
- Effectively detect and stop any discovered agent release and isolate any dissemination device(mandatory)
- Optional: Effectively package and transport a WMD agent enclosed sample container to the "Warm Zone" for decontamination and further testing.

Individual and Team Task Descriptions

Mission Objective

In this scenario the team must properly survey a location suspected of having been contaminated by an unknown HazMat/WMD agent. This training scenario provides the Hazmat Recon member with challenging tactical situations for chemical, biological, radiological or explosive agent detection and identification, and containment of dissemination devices associated with a Weapons of Mass Destruction (WMD) event.. The student is expected to operate agent detection/identification instruments and equipment, follow proper search and survey procedures as well as utilize tactical decision making during the performance of the training tasks.

The following are the tasks that must be performed by the members of the HazMat Recon team:

HazMat Recon Mission Tasks

	SubTasks	*Steps*	*SubSteps*
1	**Prepare for Hot Zone**	Don Level A PPE	
		Establish the mission abort/turn-back criteria and mission abort indicators	
		Review any structural diagrams or blueprints available	
		Select equipment and supplies for mission and perform equipment checks.	
		Activate SCBA	
		Request permission to enter Hot Zone	
		Enter Hot Zone	
2	**Search for HazMat/WMD agent**	Visually scan area to locate potential sites of HazMat/WMD agents	
		Use thermal-imaging camera (TIC) to scan walls, objects and obstructions for presence of agents.	
		Scan for presence of unknown liquids on exposed surfaces or victims	
		Report status/position to HazMat Officer at pre-determined intervals and checkpoints.	
		As each HazMat/agent is located, immediately report their location to HazMat Officer/ incident command (IC)	
3	**Search, Identify and Record Significant Environmental Features**	Use clipboard to note locations of all hazards, air vents and other major obstacles on a map of the facility	

	SubTasks	Steps	SubSteps
4	**Detect and Identify Agent**	Determine if the WMD agent is a chemical, biological, radiological or explosive agent	
		Scan area to detect any secondary device or weapon nearby, including as part of each victim	If secondary device/weapon present
			Move away from the victim and behind a barrier
			Use TIC to determine if threat has moved away.
			Advise IC immediately if any weapon detected
		Approach suspected HazMat/WMD agent	Move to HazMat/WMD agent
		Identify HazMat/WMD agent using specific instrumentation	Use specific instruments and devices
			Wait until detection/ID instrument provides a reading
			Advise IC immediately of HatMat/WMD agent ID
5	**Contain the release of an open HazMat/WMD container**	If dissemination device pressurized	Turn valve off
		If dissemination device not pressurized	
		If dissemination device potentially explosive	Evacuate area
6	**Exit the Hot Zone**	Call for clearance to move out of hot-zone into the decon area	

The following are tasks that must be performed when the HazMat Team encounters unexpected or dangerous situations:

	Situation	Task Steps	SubSteps
1	**Low Air.**	The following actions should be taken if a low-air alarm activates inside a protective suit: • Notify survey TL to initiate decontamination. • Notify your TL of the problem. • Proceed to the decontamination line as directed.	
2	**Team Member Separated**	The following actions should be taken if team members become separated (TL must advise the OPCEN of the situation.): • Proceed to the rally point. • Request a backup team if required. • Advise the OPCEN of the current situation and proceed as directed.	
3	**Team Member Down, Overheated or Injured**	The following actions should be taken if a Recon team member becomes overheated or injured: • Report situation to the Recon TL and HazMat Section Chief. • Implement the emergency plan • Standby two-member Hazmat Rescue Team will immediately enter the Hot Zone on Section Chief command, locate the stricken member and immediately evacuate both Recon entry team members to the Warm Zone for immediate decontamination and emergency medical care. • During an explosion that has also produced a chemical agent, the Recon team also could be a victim of an overhead structural collapse/debris fall/ground collapse or come into contact with exposed electrical line in contact with a broken water or natural gas line.	

	Situation	Task Steps	SubSteps
4	**Team Member Contaminated.**	The following actions should be taken if a team member becomes grossly contaminated: • Inform the OPCEN. • Conduct emergency decontamination. • Evacuate from the hot-zone immediately. • Proceed to the decontamination line immediately with a team member.	
5	**Flammable Chemicals Detected**	The following actions should be taken if chemicals are detected that are within flammability ranges: • Abort the entry and proceed to the clean area. • Notify the OPCEN, and request guidance. • Plan for mitigation, and execute on order.	
6	**Torn PPE**	Torn by brushing up against an object or by a walking victim. The following actions should be taken if a suit is torn: • Conduct emergency decontamination. • Apply tape, if possible. • Request a backup team if required. • Proceed immediately to the decontamination line with a team member. • Notify the Recon TL. • Unaffected team members await further orders before entering the decontamination area.	
7	**Terrorist Encountered**	For example, a terrorist who is still on the scene trying to prevent the team from stopping the release of a HazMat chemical agent. The following action should be taken if a terrorist is encountered: • Take cover if available. • Inform the Recon TL and HazMat Section Chief. • Move out of the area as soon as possible.	

User Interaction Models

Skill Level 1: Demonstration-based Instruction

Not applicable in this Scenario given the focus of the Prototype project

Skill Level 2: Performance-based Instruction

Not applicable in this Scenario given the focus of the Prototype project

Skill Level 3: Exercise

User Interface

The user commands the movements and actions of an avatar (i.e. virtual responder) that represents their presence in the 3D world. Users are presented with a first person (looking through a Level "A" entry suit) point of view when controlling their virtual responder. The simulation UI indicates the presence of other Hazmat Entry Team members, ambulatory and non-ambulatory victims, WMD agents, terrorist suspects, and objects within the student's immediate view. Some of these may respond to student actions while others appear as part of the background. Students operate the simulation as if they were the person performing their assigned duties as a HazMat Recon team member. Students use the mouse and/or keyboard to perform actions as directed or as needed, following the basic simulation operating procedures. The student has to command the start of an action but will not have to demonstrate physical dexterity to complete the performance step. For example, for the student to command the simulation to use a specific detection instrument, the virtual responder must right click on the suspected agent within immediate view and "click on" the appropriate pop up menu option for the detection or identification instrument to initiate the desired action. Context dependent menus will be used to simplify the user interaction. For example, a thermal-imaging camera (TIC) can be activated by right clicking on an object in the world and selecting the "Thermal Imaging Camera" option from the pop up menu. The main screen then displays the camera image which will show the presence of WMD agents and devices that are beyond your field of vision.

Instruction given to Students/Users

The simulation will have the ability to play the role of the other Hazmat team members correctly and will respond to actions prompted or commanded by the student. The simulation will also play the role of HazMat Section Officer in giving or receiving pre-scripted instructions over the radio. The simulation may intervene to present alternative corrective action considerations when an inappropriate, life threatening action is taken.

Scenario Simulation

When the mission begins with an Incident Action Briefing (IAP):

"The local 911 operator has received a call advising that several people are reported down and others are having difficulty breathing at the East Metro Train

Station. The police determined that a suspicious device was left by a passenger on one of the platforms Your job is to recon the scene and disable any device you encounter. Use your clipboard to perform the reconnaissance."..

Subtask1: Prepare for Hot Zone

The scenario assumes the Hazmat Recon Team is already outfitted in Level A Personal Protective Equipment (PPE). Responders start outside the Hot Zone entry point. Assigned equipment is in the Cart nearby. Devices and equipment include:

- Level "A" PPE
- APD2000
- Sabre 4000
- M256 Kit
- Multirae
- Radmeter
- HazmatID
- GasID
- Photo Ionization Detector (PID)
- Immuno Assay Tickets
- Dosimeter
- Draeger Tubes
- Bioseeq
- Thermal Imaging Camera (TIC)

Based on available information provided in the IAP briefing, the student must select the correct "mission critical" equipment and supplies from a pop up menu provided by right clicking on the Cart. The Recon team must check their PPE and detection devices for proper functioning at regular intervals. The Section Officer reinforces the importance of the buddy system and the importance of maintaining contact with one another while operating in the hot-zone.

At this point, the system will denotes the exact time that user went "On-Air" and a timer will start counting down. This challenges the student to remain aware of his/her remaining SCBA air amount. During the mission, the student can click on the "AIR Meter" icon in their inventory to check their remaining air amount. This will challenge the student to effectively execute their mission within a time period realizing that they must also subtract the time that it took to travel to the Hot Zone entry point, enter the Hot Zone and execute critical tasks, and still allow for remaining air supply to be replaced and complete technical decontamination and report to rehab sector for rest, fluid replacement and post-exit vital signs check by standby EMS medics.

Subtask2: Search for HazMat/WMD Agent.

HazMat Recon is conducted to obtain information by visual observation and other methods in order to confirm or deny the presence of a HazMat/WMD agent or attack.

By clicking on the keyboard A(left)/W(up)/D(right)/S(down) keys, the student can direct movement during the search process.

The student proceeds to visually scan the area for potential sites of HazMat/WMD Agents. They must search the interior and exterior spaces of the environment, including street level entrances and walkways, elevators, bathrooms, escalators, baggage claim area, gates, subway train platform, track area, train cars, etc.. The student must use the TIC to scan walls, objects or other obstructions for the presence of WMD agents that are not visible through normal vision. They must carefully observe for the presence of unknown liquid puddles, HVAC vents, unknown powders or liquids on victims and on any exposed surface and be especially alert for any suspicious pressurized cylinders in open containers on a structural surface. As each HazMat Agent is located, its position must be immediately reported to incident command (IC). The team may use a size, activity, location, unit, time, and equipment (SALUTE) report. If terrorist contact is expected, the Recon unit must remain in a covered and concealed position to minimize exposure and/or contamination. HazMat Recon may also include finding clean areas and detours around HazMat-contaminated areas.

Subtask3: Search, Identify and Record Significant Environmental Features

The HazMat Recon Team Leader will have a clipboard with a sketched layout of the incident site that may be pulled up at any time. While on the Recon mission, he or she must note the physical layout of the facility, identify the locations and general boundaries of the contaminated area, place warning markers around the contaminated area, record the position and status of casualties and the locations of other major obstacles. This clipboard is interactive and allows the student to drag and drop icons that represent the different items that must be identified.

Subtask4: Identify HazMat/WMD Agent

Based on visual observations of the environment and the physical appearance of casualties, students must determine if the HazMat/WMD agent is a chemical, biological, radiological or explosive agent. Students must then approach each WMD agent using extreme caution and scan the area to detect any secondary device, or weapon nearby, or actually as part of each WMD agent. If a weapon is detected, the student must immediately move away from the WMD agent and behind a barrier and use the TIC to determine if the WMD agent has moved (for example, if attached to a victim). The HazMat Officer and IC must immediately be advised of any weapon by right clicking on the perceived threat and selecting the "Call Bomb Squad" menu option from the pop up menu. If there are no weapons present, the student must approach each suspected HazMat/WMD agent location in order to use specific detection devices and instrumentation. The student must have already used the proper general environmental testing instrument prior to using any specific agent testing and identification instruments. If this is the case, the student then uses a specific detection or I.D. instrument to determine the type of HazMat/WMD agent or device and its intensity or concentration. As each HazMat/WMD agent is tested, an "Information Box" pops up giving the student detection or identification results for the liquid, vapor or solid material. The time to test will not be in "real time" reflecting the time that it takes to use the actual testing instrument.

Subtask5: Contain Release of Open HazMat Container

If the student detects a HazMat/WMD agent dissemination device they must try to contain and control the dissemination of the agent. The following tactics may be used. If the dissemination device is a pressurized chemical cylinder or tank, the team must turn the valve off. If the agent package is a suspected to be associated with an explosive device, the Hazmat Team should immediately evacuate the area and request the standby Bomb Squad (who are often cross trained as Hazmat Technicians) to place it into a containment vessel. Alternatively, they may place a "bomb blanket" over the package (if small) to contain a resulting explosion and minimize damage.

Subtask6: Exit the Hot Zone

The Recon team advises the operations team they are departing the incident site or they receive a command to depart the incident site from the incident command center. After concluding all survey operations, the Recon team must collect their equipment for removal from the hot-zone. In general the team departs the site via the same route used to enter the site in order to properly gauge air consumption. Entry backup teams link up (informing the operations team and decontamination team, if necessary) they have departed the incident site.

On-Air Time

The user's virtual responder on-air time will be tracked. Each Recon Team and back-up rescue team member will be equipped with a 60 minute air supply cylinder. Therefore users have 60 minutes from the time they go "On-Air" to complete the required mission tasks. However, 20 minutes must be subtracted to allow for travel to (10 min) and (10 min) from the Hot Zone. This now leaves 40 minutes, but 5 minutes must be subtracted for workload stress which increases the breathing rate and 5 minutes for decontamination. 5 minutes must also be subtracted as a safety air factor. This leaves only 25 minutes for the recon team to complete their tasks, complete decon and report to rehab.

If the user clicks the "Check Air Meter" right click menu option, a pop-up timer will display the remaining available air time remaining. The user must click the "Check AIR Meter" icon at close intervals in order to ensure that he/she has sufficient time to complete the tasks. If the user fails to complete all critical tasks within the available SCBA "air time", a PPE failure will occur. If the responder depletes their SCBA air, the screen will go Black. If the responder determines that there is insufficient air supply time remaining to complete the remaining mission task the responder can report to Decon. At this point the responder will have their air bottle changed out.

Accidents and Dangerous Situations

Simulated accidents or injuries will occur as the result of chance, student action, or inaction. For example, if the responder rubs against a protruding sharp object or an oily surface, their PPE suit will tear resulting in a PPE failure. They must then immediately use the right click "Mayday" menu option to have the stand-by Hazmat Rescue Team sent to their location.

Another example would be if a responder fails to detect the presence of a hidden secondary device in the area (using the thermal imaging camera) or that a "victim" has a secondary explosive device attached to their body, the device will explode and the screen goes black. If the user detects a secondary device in the area, or on a victim, the "Bomb Squad" right click menu option must be clicked. The user must then report immediately to Decon. The Bomb Squad then arrives and "renders safe the device." The user is given a new bottle of air and sent back into the hot zone. The user must back away from the suspected device area in order not to become a victim. If the device explodes in sight of the user, the user becomes a casualty and the screen goes black.

Duration

60 Minutes

Adjustable Scenario Parameters

Each scenario will have a set of basic parameters that a supervisor/trainer can easily adjust to change the configuration and environmental details of the training exercise.

Scenario Configuration

Environmental parameters for Scenario2 include the initial conditions of the HazMat agent release. HazMat agent parameters include the type (e.g. which chemical material to use), dissemination method (e.g. pressurized spray, spill, etc.), release point(s), quantity of material to be released, extent of contamination with respect to the release point, and present state of the HazMat agent when encountered in the Hot Zone by the HazMat team. Other parameters include:

- types and locations of secondary devices in environment
- locations and conditions of victims in environment
- placement of obstacles and obstructions
- location of smoke and fire in the environment

Accidents and Dangerous Situations

When conducting a HazMat Recon mission, team members may face a number of potentially dangerous situations that require specific, immediate action. Situations that may be configured/specified include:

- Low Air
- Team Member separated
- Team Member down, overheated or injured
- Team Member contaminated
- Torn PPE
- Flammable chemicals detected
- Terrorist encountered

Performance Assessment

The following table outlines the evaluation criteria that the application will use to determine if each task in the scenario has been completed correctly. For each incorrectly performed task points are deducted from the learner's total score.

	Task Steps	Performance Measures
1	Team maintains situational awareness	• Maintained the buddy system and checks. • Remained within line of sight of each other. • Conducted continuous hazard monitoring.
2	Team conducts mission tasks according to plan	• Detected and identified all HazMat/WMD agents that were present in environment subject to mission requirements. • Recorded all significant environmental features. • Contained release of HazMat/WMD agent
3	Team responds to unexpected situations using proper procedures	• Appropriate responses demonstrated for a variety of unexpected situations
4	Team maintains communications	• Provided situation reports (SITREPs) to HazMat section officer when appropriate.

<u>Skill level 1</u>. Not applicable. Not implemented as part of this prototype effort.

<u>Skill level 2</u>. Not applicable. Not implemented as part of this prototype effort.

<u>Skill level 3</u>. The system records student actions and provides feedback in the form of an After Action Review after all responders have left the Hot Zone. Completion of all required tasks before air runs out constitutes successful completion of the mission. Performance measures for each Recon Mission subtask are as follows:

Subtask1. Correct "mission critical" equipment identified

Subtask2. All HazMat/WMD agent locations and casualty status reported to IC.

Subtask3. Locations of all hazards, victims, and major obstacles identified and noted on clipboard before mission is completed.

Subtask4. HazMat/WMD agents identified correctly. Air quality continually monitored in hot zone to identify atmospheres that are corrosive, combustible, or oxygen-deficient or that contain radioactive or toxic substances that exceed IDLH and exposure over PELs.

Subtask5. Dissemination device properly identified and contained

SCENARIO3 – HAZMAT/WMD TRIAGE MISSION

Scenario Description

Overview

The third scenario is a modification of Scenario2 and focuses on triage as opposed to recon. In this scenario the team must triage civilians in the HotZone so they can be extricated to receive appropriate medical attention. This training scenario is intended to provide the Medical Recon member with challenging tactical consideration selections for victim search, triage, treatment and extrication of casualties resulting from a Weapons of Mass Destruction (WMD) event. The student is expected to operate casualty detection, triage, and treatment devices in the performance of training tasks.

This scenario was designed to demonstrate how a preexisting scenario can be modified and extended with additional behavior components to produce a completely different training experience. This scenario is intended for both single and multi-player modes of operation.

References

- "EMS Technician Course", Installation Preparedness (IP) Program U.S. Marine Corps, 2003
- "Use of Auto-Injectors by Civilian Emergency Medical Personnel to Treat Civilians Exposed to Nerve Agent", Chemical Stockpile Emergency Preparedness Program (CSEPP), U.S. Dept. of Energy/Martin Marietta, 1996
- "Personal Protective Equipment", Chemical Stockpile Emergency Preparedness Program (CSEPP), U.S. Dept. of Energy/Martin Marietta, 1996

Prerequisite Instruction/Knowledge

Previous completion of minimum 40 hour hazmat technician and paramedic course with additional training, proper understanding and operation of Level "A" entry suit, 1-hr. SCBA, S.T.A.R.T. triage, triage tags, thermal-imaging camera, Mark-1 auto-injector kit, NAAK auto-injector kit, Lilly Cyanide Antidote Kit, and SKED stretcher is required. Previous familiarity with the use of other emergency medical response equipment and SKED multiple victim rescue system is recommended.

Scenario Design Goals and Training Objectives

Design Goals

The intent of the simulation Scenario3 is to provide Medical HazMat responders with a working knowledge of the procedures and equipment used to properly locate, triage, treat and extricate ambulatory and non-ambulatory victims who have been exposed to WMD agent release within defined time restraints.

Training Objectives

- Effectively execute all critical mission tasks within the time restraint of the Level “A” self-contained breathing apparatus (SCBA) 1 hour air bottle (mandatory)
- Operate the thermal-imaging camera to locate casualties within the determined “Hot Zone” of the attack scene (mandatory)
- Conduct a Simple Triage And Rapid Treatment (START) of each within the determined “Hot Zone” of the attack scene (mandatory)
- Attach appropriate triage tag to victim based on provided START signs/symptoms (mandatory)
- Administer the appropriate pharmaceutical antidote and number of injections to each victim based on initial and/or changing signs and symptoms (mandatory)
- Effectively direct encountered ambulatory casualties and extrication of each non-ambulatory victim to attack scene “Warm Zone” for decontamination

Individual and Team Task Descriptions

Mission Objective

The user is a medic on a four-(4) person Medical Recon team who has been trained and equipped to Hazmat technician level. Their mission is to respond to a Hazmat incident and, as part of the Hazmat sector, immediately enter the Hot Zone, search for and locate casualties using visual sight and thermal-imaging camera, properly triage each encountered casualty, select and administer the proper pharmaceutical antidote auto-injector, re-triage if necessary, and evacuate/extricate all encountered victims in the Warm Zone for decontamination. The Medical Recon element accomplishes this at the same time that the Hazmat Entry team is conducting a recon with instrumentation to identify the contaminant.

The following are the tasks that must be performed by the members of the Medical HazMat Recon team:

HazMat Medical Recon Mission Tasks

	SubTasks	Steps	SubSteps
1	Prepare for Hot Zone	Don Level A PPE	
		Select equipment and supplies for mission	
		Activate SCBA	
		Enter Hot Zone	
2	Search for Victims	Use visual sight and operate thermal-imaging camera to locate any ambulatory and non-ambulatory casualties	
		Use TIC to scan walls, objects or other obstructions for the presence of victims not visible through normal vision.	
		As each victim is located, immediately report their location to incident command (IC)	
3	Conduct triage using S.T.A.R.T. method	Determine if victim is walking or non-walking	
		Approach each victim with extreme caution	Scan area to detect any secondary device or weapon nearby, including as part of each victim
			Move away from the victim and behind a barrier
			Use TIC to determine if the victim has moved away.
			Advise IC immediately if any weapon detected
		Begin START triage if no weapons present	Move to victim
			Determine medical signs and symptoms
			Apply START method to victim based on medical signs and symptoms.

	SubTasks	Steps	SubSteps
		Select and attach START triage tag to victim	Select and attach the RED Tag inventory icon if victim is triaged as CRITICAL.
			Select and attach the YELLOW Tag inventory icon if victim is triaged as IMMEDIATE.
			Select and attach the GREEN Tag inventory icon if victim is triaged as DELAYED
			Select and attach the BLACK Tag inventory icon if victim is triaged as EXPIRED.
4	Administer pharmaceutical antidote	Based on victim's signs, symptoms and triage classification, select the and apply appropriate pharmaceutical antidote	Choices include:
		.	ATROPINE
			2-PAM
			the NAAK
			B.A.L. (British Lewisite Antidote)
			C.A.K. icon (Cyanide Antidote Kit)
5	Evacuate or extricate or evacuate victims to Warm Zone Decon area	Walking Victims	Assemble group of walking victims
			Contact a Level "A" Medical Rescue team to escort victims to Warm Zone decontamination area
		Non-Walking victims	Contact a Level "A" Medical Rescue team to extricate victims to Warm Zone decontamination area
6	Exit the Hot Zone	Report to Decon	

The following are tasks that must be performed when the HazMat Team encounters unexpected or dangerous situations:

	Situation	Task Steps	SubSteps
1	Low Air.	The following actions should be taken if a low-air alarm activates inside a protective suit:	
		Notify survey TL to initiate decontamination.	
		Notify your TL of the problem.	
		Proceed to the decontamination line	
2	Team Member Separated	The following actions should be taken if team members become separated (TL must advise the OPCEN of the situation.):	
		Proceed to the rally point.	
		Request a backup team if required.	
		Advise the OPCEN of the current situation and proceed as directed.	
3	Team Member Down, Overheated or Injured	The following actions should be taken if a Recon team member becomes overheated or injured:	
		Report situation to the Recon TL and HazMat Section Chief.	
		Implement the emergency plan	
		Standby two-member Hazmat Rescue Team will immediately enter the Hot Zone on Section Chief command, locate the stricken member and immediately evacuate both Recon entry team members to the Warm Zone for immediate decontamination and emergency medical care.	
		During an explosion that has also produced a chemical agent, the Recon team also could be a victim of an overhead structural collapse/debris fall/ground collapse or come into contact with exposed electrical line in contact with a broken water or natural gas line.	
4	Team Member Contaminated.	The following actions should be taken if a team member becomes grossly contaminated:	

	Situation	Task Steps	SubSteps
		Inform the OPCEN.	
		Conduct emergency decontamination.	
		Evacuate from the hot-zone immediately.	
		Proceed to the decontamination line immediately with a team member.	
5	Flammable Chemicals Detected	The following actions should be taken if chemicals are detected that are within flammability ranges:	
		Abort the entry and proceed to the clean area.	
		Notify the OPCEN, and request guidance.	
		Plan for mitigation, and execute on order.	
6	Torn PPE	Torn by brushing up against an object or by a walking victim. The following actions should be taken if a suit is torn:	
		Conduct emergency decontamination.	
		Apply tape, if possible.	
		Request a backup team if required.	
		Proceed immediately to the decontamination line with a team member.	
		Notify the Recon TL.	
		Unaffected team members await further orders before entering the decontamination area.	
7	Terrorist Encountered	For example, a terrorist who is still on the scene trying to prevent the team from stopping the release of a HazMat chemical agent. The following action should be taken if a terrorist is encountered:	
		Take cover if available.	
		Inform the Recon TL and HazMat Section Chief.	
		Move out of the area as soon as possible.	

User Interaction Models

Skill Level 1: Demonstration-based Instruction

Not applicable in this Scenario given scope of Prototype project

Skill Level 2: Performance-based Instruction

Not applicable in this Scenario given scope of Prototype project

Skill Level 3: Exercise

User Interface

The user commands the movements and actions of an avatar (i.e. virtual responder) that represents their presence in the 3D world. Users are presented with a first person (looking through a Level "A" entry suit) point of view when controlling their virtual responder. The simulation UI will indicate the presence of other Hazmat Entry Team members, ambulatory and non-ambulatory victims, terrorist suspects, and objects within the student's immediate view. Some of these may respond to student actions while others will appear as part of the background. Students will operate the simulation as if they were the person performing their assigned duties as a Medical Recon element responder. Students use the mouse and/or keyboard to perform actions as directed or as needed, following the basic simulation operating procedures. The student will have to command the start of an action but will not have to demonstrate physical dexterity to complete the performance step. For example, if the student commands the simulation to attach a triage tag or administer an antidote, the virtual responder will right click on the casualty within immediate view and then "click on" the appropriate triage tag or pharmaceutical menu option from the pop up menu to administer the desired action step. The inventory pop up menu will display the number of remaining triage tags and antidote kits. Once the student starts the scenario, the screen changes to present a 3D view of the environment. By clicking on the thermo-imaging camera menu option from the right click popup menu. The main screen then displays the camera image which will show the presence of casualties that are beyond the user's field of vision, or hidden behind walls or other structures.

Instruction given to Students/Users

The simulation can play the role of the other Hazmat team members correctly and will respond to actions prompted or commanded by the student. The simulation will play the role of Incident Command in giving or receiving pre-scripted instructions over the radio. The simulation may at times intervene to present alternative corrective action considerations when an inappropriate, life threatening action is taken.

Scenario Simulation

When the mission begins the student is presented with the following Incident Action Briefing (IAP):

> *"The local 911 operator has received a call advising that several people are reported down and others are having difficulty breathing at the East Metro Train Station. The police determined that a suspicious device was left by a passenger on one of the platforms. Another team has performed a reconnaissance of the hotzone. Your job is to triage and evacuate any victims they encountered. Use your clipboard to locate the victims."..*

Subtask1: Prepare for Hot Zone

The scenario assumes the Medical Hazmat Team is already outfitted in Level A Personal Protective Equipment (PPE). Responders start outside the Hot Zone entry point. Assigned equipment is in a cart nearby. Devices and equipment include:

- Level "A" PPE
- Thermal-imaging camera
- S.T.A.R.T. triage tag kit (50 tags)
- 30 Mark-1 Antidote Auto-Injectors in a medical kit (Atropine & 2-PAM)
- 5 NAAK Auto-injectors
- 5 B.A.L. Antidote injectors

Based on available information from the IAP briefing, the student must select the correct "mission critical" equipment and supplies from the right click pop up menu.

The student must then go "On Air" in the "Cold Zone." At this point, the system will denote the exact time that user went "On-Air" and a timer will start counting down. This will challenge the student to remain aware of his/her remaining SCBA air amount. During the mission, the student can click on the "Check Air Meter" right click menu option to check their remaining air amount. This will challenge the student to effectively execute their mission within a time period realizing that the student must also subtract the time that it took to travel to the Hot Zone entry point, enter the Hot Zone and execute critical tasks, and still allow for remaining air supply to be replaced and complete technical decontamination and report to rehab sector for rest, fluid replacement and post-exit vital signs check by standby EMS medics.

Subtask2: Search for Victims

The user is presented a first person point of view for operating their virtual responder. By clicking on the keyboard A(left)/W(up)/D(right)/S(down) keys, the student can direct movement during the search process. Users must visually scan the area to locate any victims as they move the environment. The user proceeds to search the displayed walls and images of the structural interior and the outside of the structure and adjoining terrain. As the user progresses forward, he/she will hear multiple casualty voices calling out for help. The user must use the thermal imaging camera (TIC) to help locate the voices. The

TIC is used to scan walls, objects or other obstructions for the presence of victims who not visible through normal vision.

The user will also hear radio transmissions from incident command regarding additional information that the Hazmat Recon Team has provided regarding additional hazards, dissemination devices, detection instrument readings, and observed victims. As each victim is located, their position must be immediately reported to the HazMat Section Officer and/or incident command (IC) by marking it on the clipboard.

Subtask3: Conduct Triage using START method

As each victim is encountered, the user must conduct a triage using the Special Triage And Rapid Treatment (START) method. Steps include:

- Determine if the victim is walking or non-walking
- Approach each victim using extreme caution. .
- Scan the area to detect any secondary device or weapon nearby (or possibly attached to victim). If a weapon is detected, the student must immediately move away from the victim and behind a barrier and use the TIC to determine if the victim has now walked away. IC must immediately be advised of any weapon by clicking on the “Call Bomb Squad” right click popup menu option.
- If there are no weapons present, the user can move to each walking and non-walking victim.
- As each victim is approached, they cry out “I can’t see anything”, I can’t breath well!”, “My eyes are burning”, I can’t see well!”, “I think I’m going to throw up!”, “I think I just urinated in my pants!”

The student then applies the START technique to determine the triage level based on the observed medical signs and symptoms.

- Select the appropriate triage tag from the inventory window.
- Use the “Apply to selection” menu option for the RED Tag inventory item if victim is triaged as CRITICAL.
- Use the “Apply to selection” menu option for the YELLOW Tag inventory item if victim is triaged as IMMEDIATE.
- Use the “Apply to selection” menu option for the GREEN Tag inventory item if victim is triaged as DELAYED.
- Use the “Apply to selection” menu option for the BLACK Tag inventory item if victim is triaged as EXPIRED.

Subtask4: Administer pharmaceutical antidote

A medical recon team member can assess the medical status of a victim by asking questions to a conscious victim, or checking for breathing of an unconscious victim; determining level of consciousness (LOC) through questions and observations; and Squeezing victim’s fingernail to check for blood return. The spectrum of victim responses are modeled from known and documented victim responses from exposure to explosive, radiological effects, blister agent, nerve agent and other common toxic industrial chemical. If the proper pharmaceutical antidote and/or other critical medical action is administered within the proper defined time period, the victim will improve and triage status will change. If the wrong drug or dosage is administered there will be victim signs and symptoms that will indicate this which can change the outcome of the patients condition.

In this subtask the user must select and administer proper pharmaceutical antidote based on victim's signs, symptoms and triage classification. To perform these actions do the following:

- Right click on the victim and select the appropriate pharmaceutical antidote from a Diazepam, Pralidoxime, or Atropine. Auto-injector from the right click popup menu.

Subtask5: Evacuate or extricate or evacuate victims to Warm Zone Decon area

For each individual triaged walking patient, the user selects the victims "Evacuate" right click menu option to direct a patient to evacuate. For each triaged non-walking patient, the user selects the "Call Rescue Team" right click menu option to have another Level "A" Medical Rescue element team respond and extricate victim to the decontamination area.

On-Air Time

The user's virtual responder on-air time will be tracked. Each Recon Team and back-up rescue team member will be equipped with a 60 minute air supply cylinder. Therefore users have 60 minutes from the time they go "On-Air" to complete the required mission tasks. However, 20 minutes must be subtracted to allow for travel to (10 min) and (10 min) from the Hot Zone. This now leaves 40 minutes, but 5 minutes must be subtracted for workload stress which increases the breathing rate and 5 minutes for decontamination. 5 minutes must also be subtracted as a safety air factor. This leaves only 25 minutes for the recon team to complete their tasks, complete decon and report to rehab.

If the user clicks the "Check Air Meter" right click menu option, a pop-up timer will display the remaining available air in P.S.I. and time remaining. The user must click the "Check Air Meter" right click menu option at close intervals in order to ensure that he/she has sufficient time to complete the tasks. If the user fails to complete all critical tasks within the available SCBA "air time", a PPE failure will occur. If the responder depletes their SCBA air, the screen will go Black. If the responder determines that there is insufficient air supply time remaining to complete the remaining mission task the responder can report to Decon to have their air bottle changed out.

Accidents and Dangerous Situations

Simulated accidents or injuries will occur as the result of chance, student action, or inaction. For example, if the responder rubs against a protruding sharp object or an oily surface, their PPE suit will tear resulting in a PPE failure. They must then immediately select the "Mayday" right click menu option to have the stand-by Hazmat Rescue Team sent to their location. Another example would be if a responder fails to detect the presence of a hidden secondary device in the area (using the thermal imaging camera) or that a “victim” has a secondary explosive device attached to their body, the device will explode and the screen goes black. If the user detects a secondary device in the area, or on a victim, the devices “Call Bomb Squad” right click menu option must be clicked.. The Bomb Squad then arrives and “renders safe the device.” The user must back away from the suspected device area and report to decon in order not to become a victim. If the device explodes in sight of the user, the user becomes a casualty and the screen goes black.

Duration

60 Minutes

Adjustable Scenario Parameters

Scenario Configuration

Environmental parameters for Scenario3 include the initial conditions of the HazMat agent release. HazMat agent parameters include the type (e.g. which chemical material to use), dissemination method (e.g. pressurized spray, spill, etc.), release point(s), quantity of material to be released, extent of contamination with respect to the release point, and present state of the HazMat agent when encountered in the Hot Zone by the HazMat team. Other parameters include:

- types and locations of secondary devices in environment
- locations and conditions of victims in environment
- placement of obstacles and obstructions
- location of smoke and fire in the environment

Users will have a total of thirty (8) victims to locate, triage, administer any required antidotes, and either evacuate, or extricate within the time frame of air supply they have available during the scenario.

Accidents and Dangerous Situations

When conducting a HazMat Medical Recon mission, team members may face a number of potentially dangerous situations that require specific, immediate action. Situations that may be configured/specified include:

- Low Air
- Team Member separated
- Team Member down, overheated or injured
- Team Member contaminated
- Torn PPE
- Flammable chemicals detected
- Terrorist encountered

Performance Assessment

The following table outlines the evaluation criteria that the application will use to determine if each task in the scenario has been completed correctly

	Task Steps	Performance Measures
1	Team maintains situational awareness	Maintained the buddy system and checks.
		Remained within line of sight of each other.
		Conducted continuous hazard monitoring.
2	Team conducts mission tasks according to plan	Search for victims subject to mission requirements.
		Conduct triage using START method
		Administer pharmaceutical antidote
		Evacuate or extricate victims to Warm Zone
3	Team responds to unexpected situations using proper procedures	Appropriate responses demonstrated for a variety of unexpected situations
4	Team maintains communications	Provided situation reports (SITREPs) to HazMat section officer when appropriate.

Skill level 1. Not applicable. Not implemented as part of this prototype effort.

Skill level 2. Not applicable. Not implemented as part of this prototype effort.

Skill level 3. The system records student actions and provide feedback in the form of an After Action Review after all responders have left the Hot Zone. Completion of all required tasks before air runs out constitutes successful completion of the mission. For each incorrect action points are deducted from your total score.

SCORM INTEGRATION

Metatagging

The SCORM metadata specification has several tags that are useful for simulation components. These include the Format, Type, and Name tags.

The Format tag can be used to describe the type of backing asset used by a component. Values would include model, animation, terrain database, audio/wav (for sounds), image/Targa (for textures), font, and movie/AVI (for movies). The Format tag has been expanded to include types that are not MIME.

The Type and Name tags are related in use. The Type tag is used to describe the type of technology for this component. Type tag can specify that this resource is used with simulations (and possibly even be more specific and specify the component type i.e. Production, Scene, Actor, Behavior, Event, etc.). So, its value would be "simulation." The Name tag is used to describe the specific name of technology that must be used with this asset. In this case one might specify "Proscena."

For the scenarios 1, 2, and 3 we tagged all of the components but not the backing assets as they have no ability to be used in side of simulation without an accompanying component definition. Currently there is no mechanism in the SCORM Run-Time-Environment for transferring assets to the file system on a client machine to be used by an application. This is inherent in the web-based architecture of the SCORM RTE. For this reason, Meta-data tags are only useful for storing the content in SCORM repositories for search capabilities.

LMS

While no requirement was established in this prototype for creating supplementary training content to accompany the simulations developed, it was deemed necessary to demonstrate how a SCORM conformant lesson could be integrated with a produced simulation. This integration involved two tasks. The first task was the development of a piece of middleware that was capable of launching a Simulation Content Object (SCO) or production. The second task was to extract student performance data from the simulation to determine whether learning objectives were satisfied or not satisfied by the student within the simulation.

An applet was developed to launch a production from a SCO. Upon completion, the SCO-launched simulation returns data to the SCO so that the results of the training can be communicated back to the SCORM RunTime Environment. The Java Applet acts as a proxy between the SCO, which is written in JavaScript, and the launched application.

The steps of the process for launching an application from a SCO and writing the results back to the LMS , are as follows:

1. SCO launches the Java Applet.
2. Java Applet method lauches Application.
3. Application executes scenario.

4. Upon completion of the scenario, the application sends the results of the training back to the applet via the socket the applet was listening on.
5. The applet returns from the method returning the results it received.
6. The SCO processes the results returned from the applet and writes the results returned back to the LMS.

Each scenario contains a number of objectives. Upon completion of the scenario, each objective is either “satisfied” or “not satisfied”. These are the results that are returned from the application. In the example SCO created for the prototype, these results are written to local objectives which have the same names as the objectives of the scenario. A second SCO, the Training Results SCO, was created that can be used to present the results of the training stored in the global objectives.

While outside of the scope of this prototype, more novel ways of interacting with the LMS were also explored. None of these methods are SCORM conformant but they do hold promise for improved authorability in a future incarnation of SCORM. One method in particular was to actually encode the LMSGet and LMSSet functions as behavior components and have the Production manage the connection to the LMS. This frees the simulation from the constraint of the web browser allowing for better performance and a unified authoring experience. Instead of leaving the Authoring Tool all data model elements are bound to Actor Properties and syncronized with the LMS at user designated intervals. In the multi user environment of the ERT simulations the client syncronization features of HLA were exploited to share and maintain consistency amongst learner data. Seperate indepent connections to the LMS could then be opened and managed by the production allowing for multi user pedagogies to be explored.

PRODUCTIVITY

The production of simulation-based learning content using existing tools and techniques is prohibitively expensive in both time and money. High temporal and monetary costs put this technology out of the reach of organizations that would otherwise benefit from it. A majority of the high cost originates from four sources: bottlenecks caused by technical dependencies between team members, supplemental development to erect and maintain the production pipeline, failure to exploit reuse opportunities, and a lack of means to quickly and inexpensively validate design ideas.

To address these problems soVoz has developed this prototype which consists of an SDK, player and supporting authoring tools for the structured design of simulation content based on reusable components. The toolset affords dramatic time and monetary savings by enhancing an organizations existing production pipeline thereby making the construction of simulation content an option for more organizations.

This prototype leveraged a previous prototype effort by Engineering & Computer Simulations, Inc. (ECS) in the area of Civil Support Team Training (CSTT). The intention was to show that by using a component based approach, one can greatly reduce the cost of developing such solution.

The original ECS prototype produced one scenario with two skill levels. According to ECS 6.5 months or 19.5 man months were spent actually constructing the application. Using the soVoz component based approach we were able to completely replicate the ECS prototype (Scenario1) in about 1.5 months (i.e. 4.5 man months), and then produce two additional scenarios in approximately 3.5 months That is, Scenario 2 took approximately 1.8 months to develop (5.4 man months) and Scenario 3 took 1.9 months to develop (5.7 man months). In order to come to an accurate assessment of the Authoring Tools impact on the development process all work that wasn't affected by the tool was subtracted from the calculation. Examples of things subtracted from these calculations are the time it took for art construction, working with the SMEs, documentation, etc.

The soVoz toolset simplifies development and empowers all team members with a means of independently validating the efficacy of their ideas from the beginning of the development process. The toolset also provides a rallying point for artists, programmers, designers, and managers to design, construct, and test content collaboratively. Developers stand to benefit in the following ways.

Artists:

- Can integrate and tune their artwork in the runtime environment without consulting programmers.
- Can enhance their artwork by leveraging the toolset's advanced animation and simulation facilities designed specifically for real-time use.
- Can preview artwork in real-time on all target platforms.

Programmers:

- Can write reusable components for designers that they can set in relationship to one another to create new content.
- Can prototype and test components without completed artwork.
- Can customize the authoring tool to meet domain specific needs.
- Can expose unlimited power and flexibility by accessing the full complexity of the underlying game engine.
- Can edit, debug, and profile components in real-time through the authoring tool.

Designers:

- Can construct content from a set of reusable components without consulting programmers or artists.
- Can prototype design ideas ahead of the main development effort by leveraging a library of off the shelf content.
- Can identify future work for artists and programmers by using notes or placeholders.
- Can preview design changes by selectively integrating newly produced content.
- Can test the application at any stage of development on all target platforms.

Managers:

- Can reduce risk by validating design ideas early and often.
- Can speed delivery by focusing the team on content creation as opposed to technology development.
- Can eliminate bottlenecks by allowing team members to work independently of one another.
- Can make business decisions with greater ease and accuracy by producing working demonstrations earlier.
- Can reduce the expense of human resource management by eliminating dependencies on and between experts.

AUTHORABILITY

One of the objectives of this prototype was to involve subject matter experts (SMEs) in the content development process in unprecedented ways.

For Scenarios 2 and 3 we enlisted the help of EAI Corporation located in Aberdeen, Maryland who provided the necessary subject matter expertise to create an educational substantive simulation. Numerous meetings were held in the design phases to establish learning objectives and ensure a technically accurate set of materials. Additional, meetings and exchanges of ideas took place throughout the prototype effort.

Ultimately, we envision the subject matter expert being the principle author and maintainer of these new kinds of training experiences. During the course of this prototype we had to rethink the technology and our development methodology to make the SME a more integral part of the development process. We make the following recommendations for future efforts.

- ***SMEs Need More Education to Effectively Participate***. One thing that was immediately apparent when we began designing the scenarios was the complete unfamiliarity with video games that pervaded the ERT community. Most of the community has had some table top gaming experiences. However, most of the SMEs that we engaged had only very limited experience with computers and almost no direct experience with video games. Without a point of reference they were unable to articulate how to capture their training experiences in video game form. They lacked a basic understanding of game mechanics and thus were largely dependent on us to help them interpret their ideas in game form. If SMEs are to become the principle authors of simulation based training they will need a broader understanding of computers/game design. The vocabularies, methodology, and theory of game design need to be codified in a form that is accessible to this audience if they are to be effectively integrated into the development process. We recommend a crash course in game design before beginning
- ***SMEs and Teachers are not necessarily the same thing.*** Find someone with teaching experience. When the prototype began we had access to two SMEs. Circumstance led to the replacement of one of them with someone who also had experience teaching. The differences of opinion between these two individuals were so dramatic at times that we were unsure as to which direction to go in. Ultimately we decided to err on the side of the SME with teaching experience. The teacher had experience in the classroom and an intimate familiarity with the audience for these scenarios. The trainer in our opinion was better able to articulate an educational strategy as opposed to only making certain the scenarios were operational correct. We highly recommend involving SMEs that have had some teaching experience.
- ***Continuous access to SMEs is critical to creation and testing of training scenarios.*** Due to time and budgetary constraints we were unable to have continuous access to our SMEs. We had to engage them selectively at various points in the project. This lack of communication often led to misinterpretations of their ideas and the need to rework portions of the scenarios. Our expectation is that with better education most of these

problems can be ameliorated. However, we recommend that the SME regularly check the work if they are not authoring it directly.

- ***SME involvement in component design is equally as important as involvement in application design***. One mistake we made early on was to ignore the needs of the SME when building components. We built most of the components with an engineer's eye. We instinctively relied on the SMEs only to verify the final presentation. It is critical to involve the SME in the component design process so that you are guaranteed that they will understand the concepts required to manipulate it. Part of the problem stemmed from the lack of education given to the SMEs about the system. Future efforts will spend more time educating the SME about the system and making certain that the concepts incorporated into

- ***Multi-user applications complicate authoring.*** Multi-user applications complicate authoring enormously because the simulation author is forced to explicitly think about networking issues. This became a real problem for the scenarios developed for this prototype because we had to spend an enormous amount of time hiding the network infrastructure from the lay author. The networking toolkits available today require the application developer to think about networking from the get go so that data can be transferred unfettered. Moreover, they also require the user's input on a multitude of transmission issues that could be hidden completely from the user. This prototype has made great strides in making networking an afterthought but has not succeeded completely.

- ***You can't take the programming out of programming.*** Even with all the simplifications that the prototype has made to authoring simulation based training experiences there is still a requirement for logical thinking when constructing content. Some SMEs find it difficult to author even at this simplified level. More emphasis needs to be placed on fostering logical reasoning skills when training a SME to use the Authoring Tool. A tutorial that lays out best authoring practices through a demonstration would seem to be the most effective tool for accomplishing this. Through this demonstration these techniques could be highlighted and references provided for further study.

- ***The ease of use trumps flexibility***. During our testing it was evident that SMEs were willing to trade flexibility for ease of use in terms of component design. SMEs preferred to use larger do all components as opposed to arranging collections of smaller ones.

CONCLUSIONS

The ability to add simulation to instructional content can make such content more intuitive, intelligible, and effective. Current methods for producing this type of content are time consuming and require the services of professional artists and programmers, making its use by individual course instructors cost prohibitive. The Design and Development of a 3D Simulation Component Repository Supporting ADL Training and Education Applications described in this Final Report can change this by allowing content to be integrated into ADL learning applications in a cost effective manner with the following features:

	Features
Interoperability – The system shall be dissolvable into smaller pieces capable of being used in a variety of different applications.	• The component framework uses platform independent representations for components. (i.e. C++, Python, etc.) • The runtime can be hosted in a variety of different environments. i.e. Web Browser, Standalone Executable, etc.) • The authoring tool's GUI is designed modularly such that it may be integrated with other development environments.
Accessibility – The system shall employ metaphors and methodologies appropriate to the development audience.	• The framework can use industry standard representations (i.e. C++, Python, etc.) to describe components allowing them to be developed with existing tools and expertise. • Components consist of straightforward types with explicitly defined roles so that they may be understood and used by a non-expert. (e.g. Productions, Scenes, Actors, Behaviors, Events, etc.)
Reusability – The system shall produce content capable of being used again or repeatedly.	• The framework uses industry standard representations (i.e. C++, Python, etc.) to describe components allowing them to be developed with existing off-the-shelf content. • The framework is based on an open specification allowing a component to be developed with one vendor's tool and reused by another. • Components are SCORM compliant and capable of being used in existing ADL environments.
Extensibility – The system will be designed to evolve as requirements change.	• A component's description separates appearance and behavior allowing both to evolve independently. • The runtime interface is platform independent so that application content can be developed and ported to a variety of different operating systems (Windows, Unix, etc.), media layers (DirectX, OpenGL, etc.) and hardware platforms (PC, handheld devices and Game Consoles)
Affordability – The system will increase learning effectiveness while significantly decreasing development time and costs.	• The framework confines the need for professional programming and animation expertise to the component construction phase. • Components are configurable and reusable by designers and domain experts without the assistance of highly-paid professional programmers and animators. • The framework provides reuse features that allow different parts of an application to be developed in parallel rather than sequentially. • The authoring approach reduces complexity by promoting a common development model for designers, artists, programmers, and instructors. • The tool provides an interface to component repositories to facilitate reuse of components across projects and teams. • The tool reduces the risk associated with developing complex simulations by encouraging the use of proven tools and design processes. • The tool increases reliability of simulation content by allowing simulation content to be authored using proven pre-existing components as basic building blocks.

REFERENCES

1. Jones, J.L, Seiger, B.A, and Flynn, A.M. 1999. "Mobile Robots: Inspiration to Implementation," A.K. Peters, Natick, MA.

2. Dodds, P., Editor 2001. "Advanced Distributed Learning Sharable Content Object Reference Model (SCORM)" Version 1.1. Available via the Web at http://www.adlnet.org

APPENDIX A – PROSCENA DOCUMENTATION

PROSCENA SDK

COMPONENT ANATOMY

Encoding

Components can be encoded in either an object oriented script language like Python or an object oriented system language such as C++. Script languages provide a number of advantages for those looking to reduce the cost of development such as platform independence and rapid application development. System languages can be used to maximize performance by producing highly optimized platform specific code.

In the current implementation of the Component Runtime, components can be written in either Python for platform independence or C++ for exceptional performance. A unique feature of the Component Framework is the functional parity between script languages and system languages. End users can build components using either Python or C++ without making sacrifices in terms of expressivity. Script languages are not second class citizens in the Component Framework.

Composition Features

The feature set of the Component Framework focuses on providing rich metaphors for component construction and reuse.

- **Inheritance**. The framework will support the idea of reuse through inheritance. Through this concept a new type can be declared that is an extension of an existing type. This new type is said to derive from the existing type and is called a "derived" type. The original type is the new type's base type. For example, a base Actor component can be defined to represent a biped. All bipeds have a head, two arms, a torso, and two legs. That biped component can be specialized into a soldier component with the addition of features such as a gun or backpack. Such derivation provides a way of expressing an "is a" relationship. The component developer need not specify the features of the base type in a derived type. Depending on the underlying implementation languages, inheritance in the framework need not be limited to single inheritance meaning it is possible to derive a new type from multiple base types. Python components can seamlessly extend C++ components through single or multiple inheritance with full bidirectional support for polymorphism.
- **Reflection.** Components are reflective or introspective because they contain sufficient meta-data to fully describe the data they represent. This means that given the address of an component, it is possible to ask enough questions of the Runtime to fully "decode" the data the component at that address represents. Reflectivity allows the runtime to gather internal information about a component for use within the tool.

- **Mutation**. The framework exposes the ability to create new component descriptions at run time and to add new properties to existing objects. One common use of run-time field extension is when providing a feature, like physics or sound, that needs extra data (like mass or friction) added to existing components such as geometric descriptions. Using

Sovoz's mutable reflective component model, this is easy and allows existing components to be annotated with special data at will.

- **Containment**. Finally, components can be combined at run-time through containment to yield new components with different properties. For example, behaviors can contain other behaviors by designating them as sub behaviors. A behavior and its sub behaviors represent a logical unit that can be reused as a part of a larger behavior graph. For example a "Fire Prone" behavior can be created by combining a behavior that makes a soldier lay prone with a behavior that makes the soldier discharge his weapon. Libraries of complex components can therefore be built from primitive ones.

Abstract Component Types

Components are created by deriving from classes representing abstract component types. Abstract component types cannot be used in their own right as they are templates for construction of concrete components. Additionally, abstract component types can be derived into more specialized abstract component types so as to create a diverse library for component creation. Abstract component types can be distributed with the runtime or individually as needed.

All abstract component types are derived from one of the soVoz supplied abstract component base types. Each of the abstract component base types has different runtime semantics that determine the role a component will play in a VizSim application. The framework specifies three abstract component base types: Entity, Operation, and Resource.

Entity types represent mutable state in the system where as Behavior types represent manipulations or tests of that state. Resource types typically represent immutable state that can be created once and referenced many times throughout a simulation.

The types Production, Scene, Actor and any types derived from them are considered Entity types. Entity types can maintain state throughout their lifetimes and should be used as mutable namespaces for storing information in a simulation.

The types Behavior, Event and any types derived from them are considered Operation types. Operation types manipulate Entity types but maintain no persistent state of their own.

The types Camera, Light, Line, Mesh, Group, Keyboard, Sound, Joystick, Shader, Texture, and Animation and any types derived from them are considered Resource types. Resource types are used by other components to specify their inner workings.

The distinction between Entity, Operation, and Resource types is important as a component's make up and handling by the runtime is determined by their membership in one of these categories. Behaviors and Events are executed and should be implemented as stateless. All state used by a Behavior or Event should be passed to it at execution time as a parameter. The runtime can achieve great scalability by using various techniques that rely on Operation types being stateless. Likewise, the runtime can utilize other techniques to optimize the handling of Entity and Resource types.

The framework also provides several derived abstract types to aid in application construction.

Deriving from **Entity**:

- **Production -** Represents a collection of scenes which when combined form a simulation. Scenes can be pushed into and popped out of productions. By calling a production's Sense(), Control(), and Act() functions (See Section the Proscena Component Runtime for more details), the associated functions of the scene within that production are called in turn. Being virtual, these and other functions can be overridden by a derived production class.
- **Scene** – Represents a 3D space where an author constructs a portion of a Production. A Scene is a container class for Actors. Actors can be pushed into and popped out of scenes. By calling a Scene's Sense(), Control(), and Act() functions, the associated functions of the entities within that scene are called in turn. Being virtual, these and other functions can be overridden by a derived scene class. The Focus() function of an entity can be used to connect its behaviors with other entities it was designed to interact with, and the scene facilitates this joining. For example, an actor can be told to "shake the hand of your nearest neighbor". The actor can query the scene for its list of actors in order to satisfy this request.
- **Actor -** Represents the state, geometry, appearance, kinematics, dynamics, etc. of an articulated entity in a scene. Two important classes that it contains are an instance of Body and an instance of BehaviorList, which maintain the actor's body and behaviors, respectively. As the actor is incremented in time within an application, the Actor's Sense(), Control(), Act(), and Render() functions are called in order. They, in turn, call the Act() function of its Body and the Sense(), Control(), and Act() functions of its BehaviorList. Active behaviors that have implemented a Sense(), Control(), or Act() function thereby get called in sequence.

Deriving from **Operation**:

- ***Behavior*** – Represents parameterized operations on the state of an Entity in a simulation. A behavior is owned by an entity, and it represents a function that operates on its owner. Usually, low-level behaviors orchestrate the changing of active contact points or synergy groups and the movement of synergy goals. Higher-level behaviors are built up from sub-behaviors. The main functions of the base behavior class are Go(), Stop(), and IsDone(). A simulation has three main phases: a sense phase, a control phase, and an act phase. Consequently, Behavior also has virtual Sense(), Control(), and Act() functions that can be optionally overridden. These will be called automatically as needed if a behavior is active. Each entity has one instance of behavior list, and behaviors are registered with this list. During the sense, control, and act phases of a simulation, a entity's behavior list will be checked to see which behaviors are active and have a registered Sense(), Control(), or Act() function. These functions will be called in the order in which the behavior became active (with the Go() call).

  Deriving from **Behavior**:

  - ***MetaBehavior*** – The top level behavior of an entity. All entities are required to have a Meta behavior.

- ***Event*** – Represents parameterized tests and notifications for something of interest in a simulation. Events are derived from the class Event. An event is owned by a behavior. In the sense phase of a simulation all events are checked for their truth value. If an event returns true from its Test() function the owning behavior is signaled.

  Deriving from **Event**:

  - ***Behavior Status*** **-** Signals the state of a behavior in its execution cycle.
  - ***Event Status*** **–** Signals the state of an event in its test cycle.
  - ***Collision*** – Signals the intersection of two bounding areas.
  - ***Animation*** – Signals that a designated animation key has been played.
  - ***Timer*** – Signals an expired duration or regular interval.
  - ***Keyboard*** – Signals the manipulation of the keyboard.
  - ***Mouse*** – Signals the manipulation of the mouse.
  - ***Joystick*** – Signals the manipulation of the joystick.
  - ***Relational*** – Signals that a relational test of two attributes is true.
  - **Ray Cast -** Signals that a ray cast has intersected an object.
  - **Scripted –** Signals that an arbitrary operation has been evaluated.
  - **Boolean –** Signals an automatic True or False result.

Deriving from **Resource**:

- **Text -** Pixel accurate on-screen text. Text relies on a single texture that contains characters to be used for the source font. Applications can replace this texture as needed. In doing so, they will also need to define pixel-based spacing metrics for the characters texture.
- ***Camera* -** Represents a view point into a scene. A Scene may have multiple Cameras positioned throughout.
- ***Light*** – Provides illumination in a scene. There are four types of Lights: point, spot, directional, and ambient. Lights can be colored and react to an objects's material color.
- ***Line*** – Represents a line in three-dimensional space.
- ***Mesh*** – Represents a three-dimensional mesh of polygons. Meshes are the basic building blocks used to represent actors in a simulation. A collection of meshes can be designated as the body parts of an actor and animated individually or a single mesh can be designated as the body of an actor and animated by applying deformations to it.
- **Shader** – Represents a small program which processes pixels or vertexes and executes on the Graphics Processing Unit. A shader can produce meshes that appear to have fur, look like cartoons, appear to shimmer, etc.
- ***Sound* –** Represents an audio clip.
- ***Texture* –** Used to map an image or a movie onto a mesh. Any image or movie file that can be converted into a supported format can be used as a texture. The process of applying textures is like projecting an image onto an object; by default a texture is stretched or shrunk to cover an entire mesh, regardless of size or proportion. Tiling, texture size and texture projection can be used to adjust how the texture covers a mesh.
- ***Material*** - Defines a mesh’s color, shininess, glow, and transparency. A rendering style can be applied by choosing a shader to alter how the material appears on a mesh. Separate colors can be used to accentuate the highlight, ambience and glow of a mesh. A material can contain a image or a movie textured on an mesh that can play even as the mesh is transformed. It is also possible to specify how the texture should blend with the mesh's material color. A texture is tied to a material. When a texture is added to a material, all objects linked to that material are also updated.
- ***Keyboard*** – Represents the keyboard.
- ***Joystick*** – Represents the joystick.
- ***Mouse*** – Represents the mouse.

The collection above of abstract base types can be used to build a library of components for the construction of complex content.

Support Classes

Support Classes are used by a component to specify its inner workings. Each component has a unique set of associated classes that range in use from specifying internal state to referencing other components. For instance, Actor components need a means of describing their body. The Body class is used to help an actor specify the state of its body parts.

The following is a list of Support Classes and the components that use them:

Used with **All**:

- **Variant** – Complex data type that encapsulates all elemental data types used in the system ensuring their proper handling. See the Data Types section for details on the values a Variant may hold.

Used with **Entities** and **Resources:**

- ***Property* –** A name-value pair for the representation of persistent state in the system. Properties are intended for use with Entity Types. Entity types are intended to be used as repositories for persistent state in a simulation. The Property is used to specify the state of an Entity type and the rules for its use. Properties remain in memory during a component's lifetime. Properties can be initialized directly or to default values when a component is constructed

Used with **Operations**:

- ***Parameter* –** A name-value pair that represents transient input or output. Parameter is intended for use with Operation Types. Behaviors are intended to be general resources that can be applied to a wide class of entities. Behavior parameters represent the "knobs" that can be used to customize a behavior for a particular entity. Events also use parameters to customize testing. The Parameter class formalizes the concept of behavior and event parameters.

Used with **Actors**:

- ***Body*** – Body represents an actor's physical makeup. It is a container class for Body Parts and Joints. Body utilities include setting the active contact point or synergy group, accessing joints and body parts, and updating the current posture given current relative joint angles.
- ***Joint* –** It has parameters typical for a joint, such as a position and orientation relative to a parent, an Euler angle orientation order, limits of rotation, a mass, and the name of the body parts attached to the joint. It is also a container class for instances of other classes that represent synergies, such as Synergy and SynGroup.
- ***Bone* –** Bones are the rigid extensions of joints. Joints are the building blocks of skeletons. Each joint can have one or more bones attached to it. The action of a bone attached to a joint is controlled by the joint's rotation and movement.

- ***Synergy*** – Represents the mechanism that maintains goal-based constraints associated with a specific set of joints. A synergy includes a goal that holds a desired value, and a control point that represents an "actual" value. Together, the desired and actual values of a synergy define an error to be reduced. For example, the goal of a biped actor's right hand positioning synergy represents where the hand should be, and the control point is the actor's right hand joint. The inverse kinematic effect of the synergy is to reduce the error between the actual and desired hand position. Synergies also contain information regarding the goal's frame of reference, the strength of the synergy relative to other active synergies, and a list of (joint gain) pairs (called SynJoints) that denote how much each joint should contribute to the achievement of the synergy's goal. Synergies are defined with respect to a contact point, so the aforementioned actor would have a right hand positioning synergy for when it is standing on its left foot, and a different right hand positioning synergy when it is standing on its right foot. This is why Joint is a container of Synergy and SynGroup.
- **SynJoint -** A SynJoint represents how much a given joint contributes to the synergy to which it belongs. It contains a pointer to a joint and a scalar gain. The gain can be varied to influence motion resulting from a synergy. For example, consider an actor's right hand positioning synergy. The result of pulling around the actor's right hand with the elbow gain low and the shoulder gain high is quite different than with the elbow gain high and the shoulder gain low. These values can be changed at run-time to produce context-dependent behavior.
- **SynGroup -** Represents collections of goal-based constraints applied to an actor's body. Synergies are activated by group. SynGroup contains pointers to synergies, and is contained by Joint. The default biped, for example, has default synergy groups defined for each of its feet. When the actor walks, the active syngroup alternates between the right foot default syngroup and the left foot default syngroup. A syngroup can contain many synergies. For the standard biped, they include position and alignment of the hands, position and alignment of the feet, position and alignment of the head, alignment of the pelvis, center-of-gravity positioning (balance), and posture regulation.
- ***Goal*** - Represents the goal value of a synergy. For a positioning synergy, it represents the desired position (x, y, z) of the control point (a joint). For an alignment synergy, it holds the desired orientation (xa, ya, za). For the balance synergy, it represents the desired position of the actor's center-of-gravity (cg). In each case, the goal has a frame of reference; a specified joint that the goal values are referenced to. For biped positioning and alignment, for example, the goal may be relative to the active contact point (usually a foot), or some other convenient joint such as the head. The balance synergy goal is usually referenced to the active contact point.

Used with **Behaviors**:

- ***Link* -** Represents a potential transition between behavior pairs in a behavior graph.

Used with **Events**:

- ***Event Expression*** – Expression for the evaluation of starting and/ or ending events. Expressions can be parenthesized and utilize the AND, OR, and NOT operators.

Summary of Class Containment

Show below is a summary of class containment within the Component Framework.

```
Production
    Property(s)
    Scene(s)
        Property(s)
            Actor(s)
            Property(s)
                Body
                    Bone(s)
                    Joint(s)
                            Goal
                            Synergy(s)
                                pointer to Goal
                                SynJoint(s)
                                    pointer to Joint
                                SynGroup(s)
                                          pointer to Synergy(s)
`                           Camera(s)
                            Light(s)
                            Line(s)
                            Mesh(s)
                                Material(s)
                                Texture(s)
                            Shader(s)
            Text(s)
           Sound(s)
            Keyboard(s)
            Mouse(s)
            Joystick(s)
            Behavior(s)
                Parameter(s)
                Link(s)
                Events(s)
                    Parameter(s)
```

Data Types

Properties and parameters of components are composed of elemental data types.

- **Boolean -** The Boolean data type specifies a single Boolean value. Each Boolean value represents either TRUE or FALSE. How these values are represented is encoding dependent. The initial value of an uninitialized Boolean data type is FALSE.
- **Double -** The Double data type specifies one high-precision floating point number. Doubles are serialized as specified in the respective encoding. Implementation of these data types is targeted at the double precision floating point capabilities of processors. However, it is allowable to implement this data type using fixed point numbering provided at least 14 decimal digits of precision are maintained and that exponents have range of at least [-12, 12] for both positive and negative numbers. The initial value of an uninitialized Double data type is 0.0.
- **Float -** The Float data type specifies one single-precision floating point number. Floats are serialized as specified in the respective encoding. Implementation of these data types is targeted at the single precision floating point capabilities of processors. However, it is allowable to implement this data type using fixed point numbering provided at least 6 decimal digits of precision are maintained and that exponents have range of at least [-12, 12] for both positive and negative numbers. The initial value of an uninitialized Float data type is 0.0.
- **Integer -** The Integer data type specifies one 32-bit integer. Integer data types are signed. The initial value of an uninitialized Integer data type is 0.
- **String -** The String and Strings data types contain strings encoded with the UTF-8 universal character set. String specifies a single string. Strings are specified as a sequence of UTF-8 octets. Any characters (including linefeeds and '#') may appear within the string. The initial value of an uninitialized String is the empty string.

Serialization

In order to support the resumption of a simulation from a saved point the simulation may interrupt the execution of a production at anytime and instruct all components to serialize their current state. Upon the completion of the current cycle all components are instructed to persist themselves to a stream provided by the simulation. Any other inter-component information is also saved into that stream. Upon restart of the production the simulation reconstructs the production as it was at the time it was saved. All components are required to implement a Serialize () function so that they may participate in serialization. Properties and Parameters will be serialized automatically so developers are encouraged to store their data there. Any additional state must be added to the stream manually.

Standard Units and Coordinate Systems

Proscena defines the unit of measure of the world coordinate system to be meters. All other coordinate systems are built from transformations based from the world coordinate system. The following table lists standard units for Proscena.

Standard units

Category	Unit
Linear distance	Meters
Angles	Radians
Time	Seconds
Color	RGB ([0,1], [0,1], [0,1])

The framework uses a Cartesian, right-handed, three-dimensional coordinate system. By default, the viewer is on the Z-axis looking down the -Z-axis toward the origin with +X to the right and +Y straight up.

Registration

The Component Registry is a database that contains information about the components residing on a user's machine. Before a component is used it must be registered in the Component Registry. The registry stores information about a component and the assets that it references so that they may be collected for instantiation.

Registry Maintenance

The Component Registry is the application database for the component framework. The registry maintains information about all the components installed in the system. The simulation searches the registry to fulfill component creation requests. When an production requests an instance of a component, the registry is consulted to resolve the component ID with its implementation in the local file system. The information placed in the registry is gleaned from the component definition and its associated alias file. Components can be register either from the Repository Window of the authoring tool or using the command line utility <InstallLocation>\Bin\SzRegister.exe. Use the -help option for usage information.

COMPONENT EXECUTION

Simulation Loop

ProScena executes a production by looping over its members and calling various functions repeatedly. The overall simulation processing loop involves first waiting for the sample rate clock to tick. Each active entity then executes the three basic functions shown below:

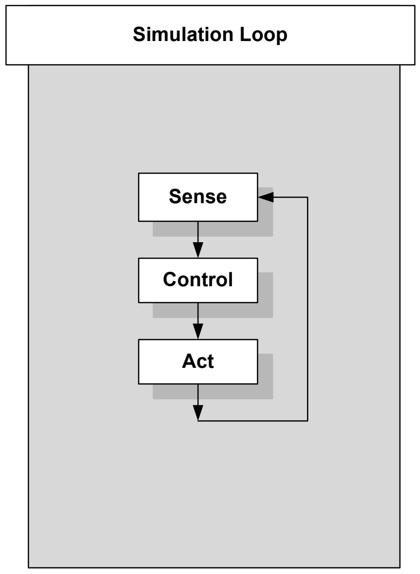


Figure 2. Phases of Execution

- **Sense.** The Sense function performs state estimation. It lets the entity know what's happening around it. The Sense function enables entities to focus on objects of interest in the world and read any virtual sensors that are active in order to determine the state of the world.
- **Control.** The Control function computes commands for the entity such as speed and direction commands for a walk behavior or the desired locations of an the actor's hands and feet. The control function represents the entity's decision-making process. The Control function permits the computation of goal-directed responses to the state of the world.
- **Act.** The Act function increments the state of the entity (position, orientation, velocity, posture, etc.)

Note the order of processing: all entities Sense, then all entities Control, then all entities Act, etc. This keeps entity decision-making synchronized, ensuring that no entity has an advantage just because it's first or last in some internal list of active entities.

Behavior System

A behavior is a hierarchical goal-directed, parameterized finite state machine At the lowest level, behaviors supply a wrapper for individual animation sequences that provides them with starting events and ending events (for triggering movements on and off), a list of parameters (speed, direction, etc.) and an execution state (idle, active, done, etc). At the mid-level, behaviors can also supply lower-level behaviors and synergies with goal values (such as the desired positions of the hands, head, feet and center-of-gravity as a function of time), set parameters, trigger subsequences on and off, and communicate with the various User Interface Subsystems (Graphics, Sound, Device I/O, Text-to-Speech, Voice Recognition, and Image Processing).

Behaviors simplify the job of animating an Actor considerably by enabling movements such as walking, running, jumping, reaching, dodging, gesturing, etc., to be parameterized and made goal-directed.

More complex animation and motion sequences can be created by “directing” an actor using behaviors accessed from the Component Registry. Direction is comprised of hierarchical groups of behaviors that are executed in series and in parallel to create an animated actor that can synthesize its movements in real-time based upon interactions with the user and the environment in which it lives, along with user-defined functions and other conditional logic For example, the task “open the door” can be created by constructing a sequence of goal-directed behaviors such as “walk”, “reach”, “grab” and “pull” with starting and ending events containing appropriate conditional logic.

Behavior Representation

Every actor has a special type of behavior called a MetaBehavior. When the Behavior Engine executes an actor’s MetaBehavior, the actor comes to life. All other behaviors associated with a character are children (sub-behaviors) of the actor/MetaBehavior. These behaviors are also hierarchical in that they can contain further sub-behaviors.

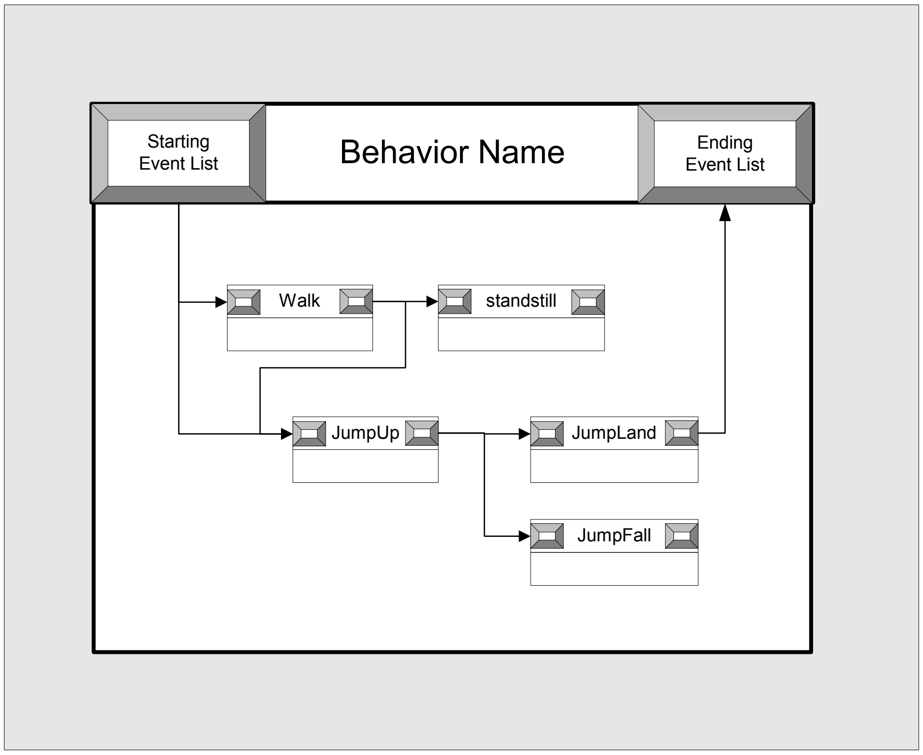


Figure 4. Typical Hierarchical Behavior. The lines connecting pairs of behavior blocks represent behavior dependencies and the buttons in the upper left and right hand corners the boolean events that cause one behavior to terminate and another to begin.

Behavior Events

Events are used to trigger or affect the sequence of actions an actor performs during the execution of a behavior. Events can be generated based on the system clock, by system events, through a script, program logic or by external devices. Starting and Ending Events are represented in the figure above as (green and red) buttons in the upper left and right corners of each block.

- *Starting Event* - a Boolean expression expressed as a script that when evaluated to true causes the behavior to begin execution.
- *Ending Event* - a Boolean expression expressed as a script that when evaluated to true causes the behavior to terminate execution.
- *Temporal Events* - a list of temporal cues relative to the behavior's start time that when evaluated to true causes the behavior to send messages to the behavior event handler to trigger various actions such as voice and audio playback, graphical special effects and system commands such as mouse clicks, key presses and standard Windows messages.

Routing

For data flow operations, each behavior has associated with it a set of parameters. Other properties and parameters can be sent to and from these parameters to create a Route. Routes can be created programmatically but for the most part they are intended as the means by which components are connected to one another from a visual editing tool. There are four basic types of routes each route is described below.

Target By Value Once	Routes the value a single time when the behavior becomes active.
Target By Value Continuously	Routes the value every frame beginning when the behavior starts and ending when the behavior stops. Changes made to the value by the behavior are not reflected in the source property.
Target By Reference	Routes the value every frame beginning when the behavior starts and ending when the behavior stops. Changes made to the value by the behavior are reflected in the source property.
Immediate By Value	Routes the value
Immediate By Reference	

GETTING STARTED

The <InstallLocation>\SDK\Solutions directory of the installation contains two Visual Studio solutions with projects that have example components. Templates.sln contains a minimalist project that can be used as a template for writing components in C++. The Samples.sln contains the projects and source code for the three component libraries provided with the authoring tool. The Component Reference in the Appendix provides details on the use of these components. These components libraries were used in implementing the example productions provided in the <InstallLocation>\Productions directory. The <InstallLocation>\SDK\Scripts directory contains two example scripts that show how to write behaviors and events in Python. For more information on scripting behaviors and events check out the Scripting Reference in the Appendix. For information on scripting actors checkout the .Act and .Bod Appendix.Each directory in the installation contains a readme.txt file that provides descriptions and instructions for the use of its contents.

PROSCENA STUDIO™

OVERVIEW

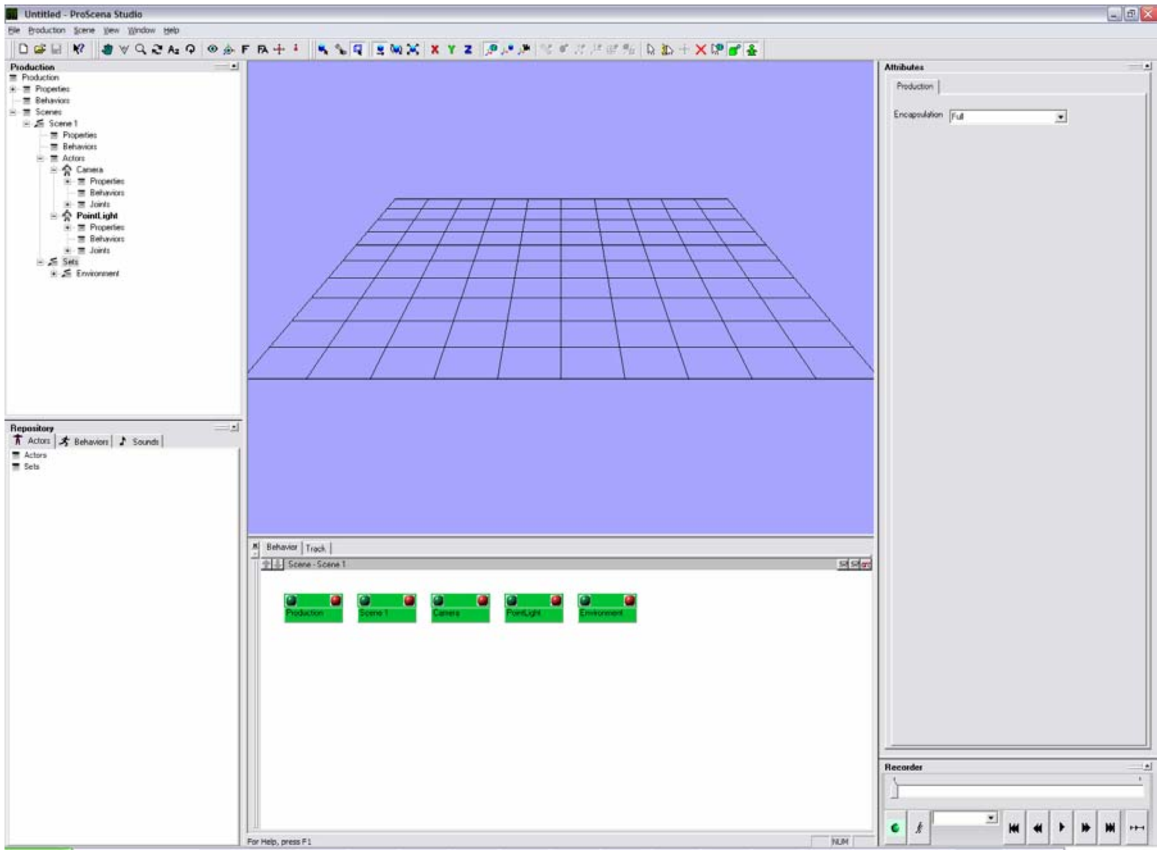
Proscena Studio allows the content expert to approach the creation of simulation content from the perspective of a "Director" working within the following paradigm:

> *All simulation content is composed of reusable components. The author considers himself in the position of the "Director" and wants a particular scene acted out. He thinks first in terms of the goals he wants the Entities to achieve and the sequence of tasks that they must perform in order to achieve these goals. As he proceeds with the design he begins to select entities and invokes relevant goal-directed behaviors; e.g. move to the (x,y,z) location I am pointing to, pick up part P, attach the part to the object I have selected, etc. to implement the subtasks. The author assumes that the components can do what he asks of them and is prepared to instruct them with any portion of the task that they cannot accomplish by themselves. Eventually the tasks are broken down into simpler and more specific subtasks as he determines the ability or lack of ability of the components he uses. In the end he may have to specify exactly how each object should move in order to accomplish the task he's instructed them to perform.*

The system described above leads the author to design simultaneously from the top down and bottom up to produce the desired animation. The system also is designed to allow for refinement and trial execution of any portion of the production as the design proceeds. This means that a component can be interactively tested once its geometry, joint hierarchy and behaviors have been defined. The parameterized nature of soVoz behaviors allows the resulting task animations to be easily modified. As a result, as tasks are defined and implemented they can be repeatedly refined until satisfactory results are achieved. In addition, animation cues that have originally been defined as explicit functions of time can be easily exchanged for cues that are event driven. For example, instead of reaching out to grab the part at frame 24, an Actor can be made to start reaching when a proximity detector determines it is within one foot of the part. This allows the task animations to be generalized and reused in different contexts.

THE USER INTERFACE

Main Window



The Main Window is the starting point for all activities within the authoring tool. The Main Window is used to maintain global settings for the production as well as orchestrate all of the unique views onto a production.

All authoring begins by either creating a new production or opening an existing one from the File Menu. The File Menu also has commands for saving a production to the file system and deploying it to a remote server or console. Additionally, the Settings\Default Search Paths menu option launches the Search Paths dialog that allows the user to change the default paths used by the system to locate components for a newly created production. These paths will persist from authoring session to authoring session. Note that changes to these paths are not reflected in a production once the production has been created. Use the Production\Settings\Search Paths menu option to affect changes to a production’s path.

The Production Menu contains commands for managing the current production. The Settings\Search Paths menu option launches the Search Paths dialog that allows the user to change the paths used by the system to locate components for the current production. Any path changes made will be saved in to the production and used on load. The

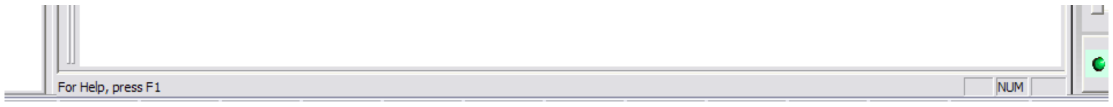
The Scene Menu has options for configuring the currently active scene. The Renderer menu option contains submenus for changing the shading, anti-aliasing, and full screen options of the Renderer.

The View Menu houses commands to open/close and refresh child windows that can be used to uniquely explore and manipulate a Production. A user can arrange the Main Window's child windows any way he or she pleases. The Main Window will save and restore a child window's configuration each time they are open and closed. User defined layouts are also preserved between user sessions.

The Window Menu controls visibility of various features of the Main Window such as the Toolbars and Status Bar. Toolbars contain a user-customizable set of short cuts for accessing commands. The Status Bar provides real-time state information about activities transpiring in the tool as well as providing direction on the use of the currently selected command.

The Help Menu provides entry to the help system, information about Sovoz, and version information about the authoring tool. Detailed context sensitive help is available anywhere in the tool by pressing F1.

Status Bar


For Help, press F1

The Status Bar, at the bottom of the screen, provides ongoing feedback, based on the current cursor position and program activity. The Status Bar also displays descriptions of commands as your cursor passes over toolbar icons or menus. When a tool or manipulator is in use, step by step instructions on how to use it are displayed on the Status Bar.

Menus

File Menu

New	Creates a new Production.
Open	Opens an existing Production.
Close	Closes the current Production.
Save	Saves the current Production in a *.szp* file.
Save As	Saves the current Production with a new name.
Settings\Default Search Paths	Opens the Default Search Paths Dialog.
Settings\Input Devices	Opens the Input Devices Dialog
Exit	Closes the current Production and exits the application.

Production Menu

Settings\Search Paths	Opens the Search Paths Dialog

Scene Menu

Renderer\ Set Shading Mode	Sets the scenes shading as either flat or smooth and either point, wireframe, or solid.
Renderer\Set Antialiasing Mode	Sets the scenes antialiasing from None up to 16x.
Renderer\ Set Full Screen Mode	Displays the scene in one of the full screen modes available on the system. Press Ctrl-Shift-X to return to windowed mode.
Show Grid	Toggles the reference grid on and off.

View Menu

Open Window	Creates a new window of the type selected in the

	cascading menu.
Refresh	Refreshes any open child windows of the Main Window.

Window Menu

Toolbar	Opens the Toolbar Configuration Dialog.
Status Bar	Toggles the visibility of the Status Bar.
Customize	Opens the Toolbar Customization Dialog.

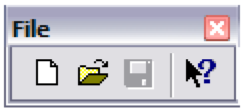
Help Menu

Contents	Opens the help system to the Table of Contents page.
Index	Opens the help system to the Index page.
Search	Opens the help system to the Search page.
Sovoz on the Web	Opens a web browser to the Sovoz corporate web page.
About Proscena	Opens the About dialog.

ToolBars

Toolbars contain short cuts for accessing commands for the various windows. Toolbars can be customized to contain a variety of different commands for optimizing workflow. By default three tool bars are available: the File tool bar which contains short cuts for managing the production, the Camera tool bar which has controls for manipulating the camera in the Scene Window, and the Picking tool bar which possesses controls for picking in the Scene Window. The user can customize the available toolbars from the Window\Customize menu option. Toolbars can be docked and undocked just like any other window.

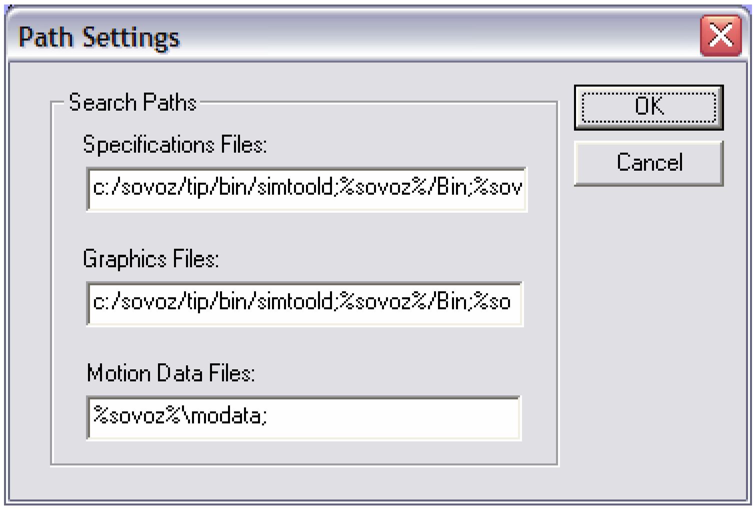
File Toolbar



Buttons are documented from left to right.

New Production	Creates a new production
Open Production	Opens a new production from the file system.
Save Production	Save the current production.
Launch Help	Launches the help system.

Default Search Paths Dialog



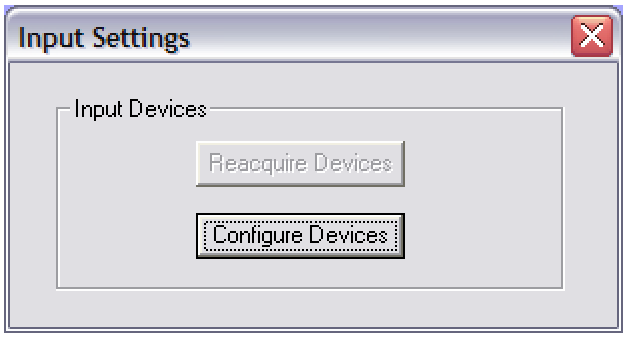
Specifications Files: c:/sovoz/tip/bin/simtoold;%sovoz%/Bin;%sov

Graphics Files: c:/sovoz/tip/bin/simtoold;%sovoz%/Bin;%so

Motion Data Files: %sovoz%\modata;

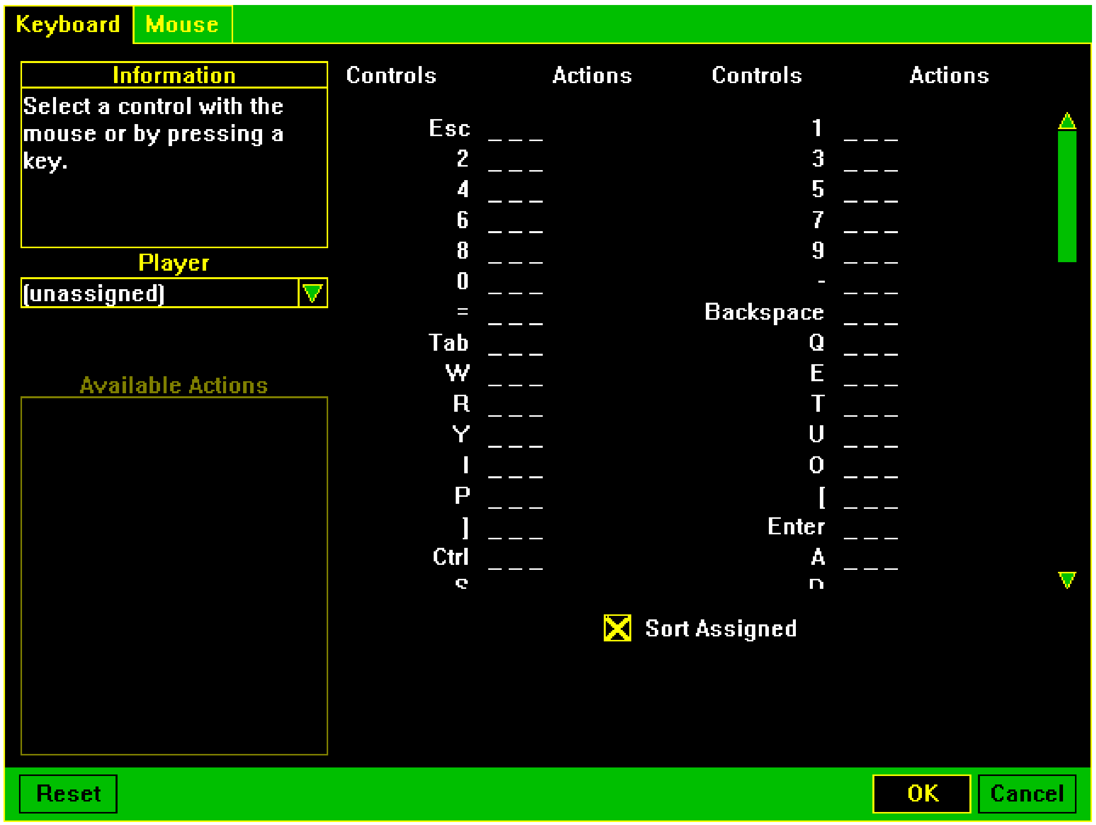
The Default Search Paths Dialog is accessed from the File\Settings\Default Search Paths menu option. This dialog allows the user to specify the default paths for finding components. These paths are used when a new production is created. Changes to these paths are not reflected in any production after it has been created. Use the Production\Search Paths menu option to access the Search Paths dialog which changes the current production's paths.

Input Devices Dialog



The Input Devices Dialog allows the user to configure input devices attached to the system. The dialog provides facilities for remapping devices as well as reacquiring devices that may have been recently installed or reinstalled. Pressing the Configure Devices button opens the Mapping Dialog. Pressing the Reacquire Devices button detects newly added devices.

The Device Mapping Dialog



Information

Select a control with the mouse or by pressing a key.

Player

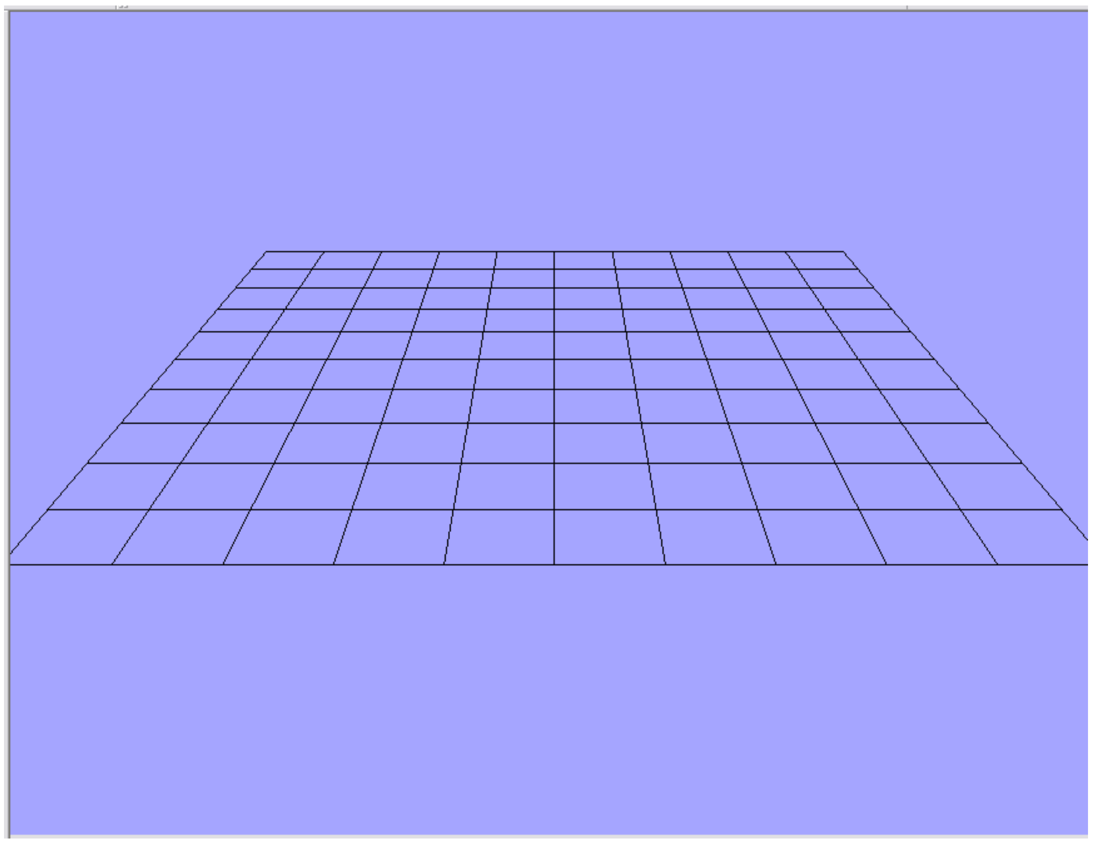
(unassigned)

Available Actions

Controls	Actions	Controls	Actions
Esc	___	1	___
2	___	3	___
4	___	5	___
6	___	7	___
8	___	9	___
0	___	-	___
=	___	Backspace	___
Tab	___	Q	___
W	___	E	___
R	___	T	___
Y	___	U	___
I	___	O	___
P	___	[	___
]	___	Enter	___
Ctrl	___	A	___
S		D	

[✓] Sort Assigned

Scene Window



The Scene Window displays a three dimensional rendering of the current scene. The Scene Window can selectively display a scene's entities and resources from a variety of different perspectives to aid in simulation construction.

The view in the Scene Window is linked to a camera that "looks" at the scene. The camera's position, orientation, and attributes determine what you see through that particular view. The Scene Window offers an assortment of navigation tools that can be used to create new views.

The actors comprising the active scene can be manipulated directly from the Scene Window. Actors can be selected for manipulation by clicking on them directly or by lassoing them. Selected actors can be manipulated with a variety of tools. Each tool provides a unique manipulator for changing the properties of an object. Any changes applied in the Scene Window can be undone, redone, and repeated as necessary.

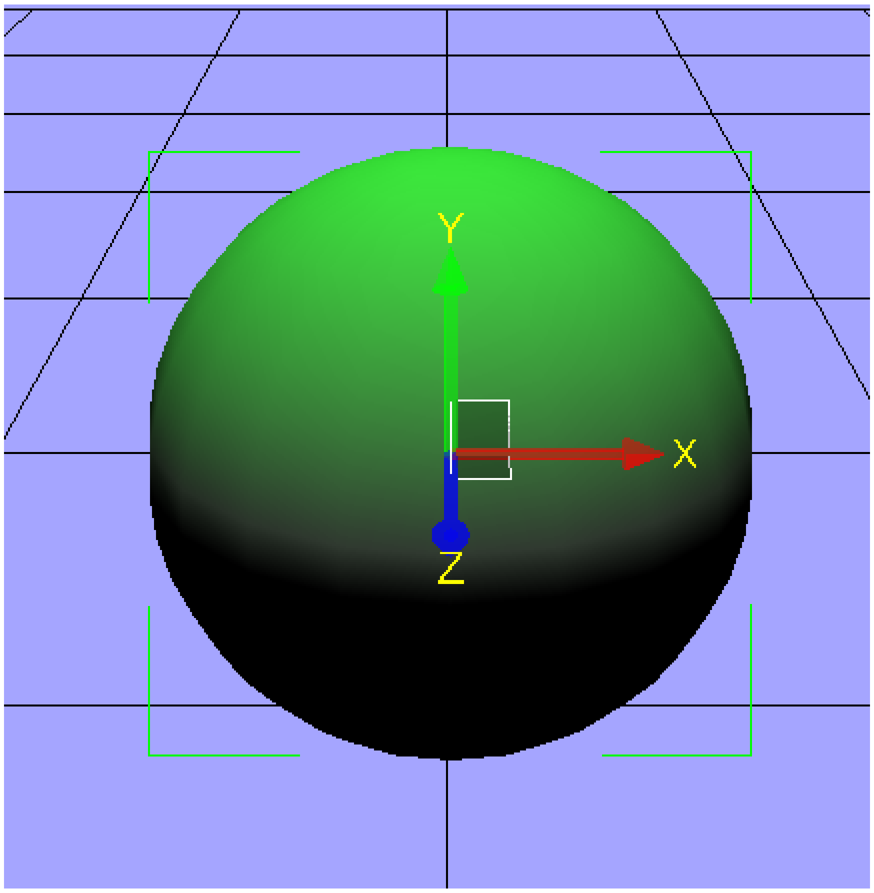
The way in which the Scene Window renders the scene can be changed from numerous points. These rendering modes range in use from providing additional information to the user to speeding rendering. The Scene\Renderer\Set Shading Mode menu on the Main Window provides global options for smooth or flat shading a wireframe, point, or solid representation of an object. Additionally, there are options that work in conjunction with the major shading modes such as Fog and Background color that can be accessed through the Environment Actor's properties. The Main Window's Scene\Renderer\Set Antialiasing Mode menu options allow the user to regulate the amount of antialiasing applied to the scene. The

Scene\Renderer\Set Full Screen Mode menu allows the user to change whether the Production is viewed in Full Screen as well as the resolution at which to display it.

The scene can be populated by dragging actors from the Repository Window on to the Scene Window. When dropped into a scene an actor appears at the origin.

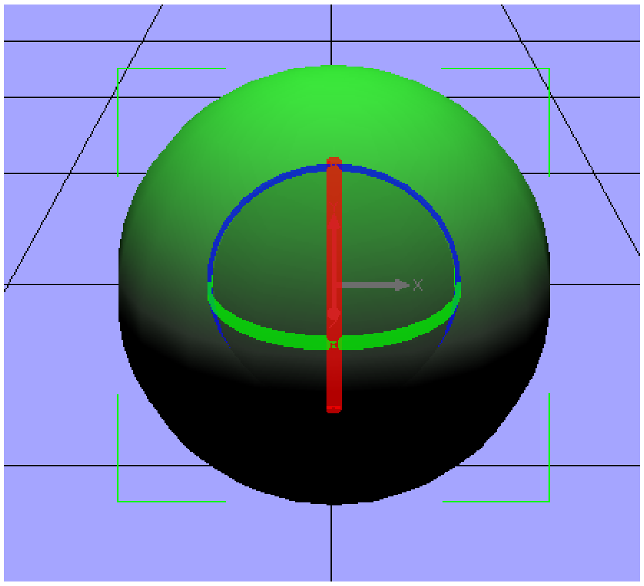
Manipulators

<u>**Transform Manipulator**</u>



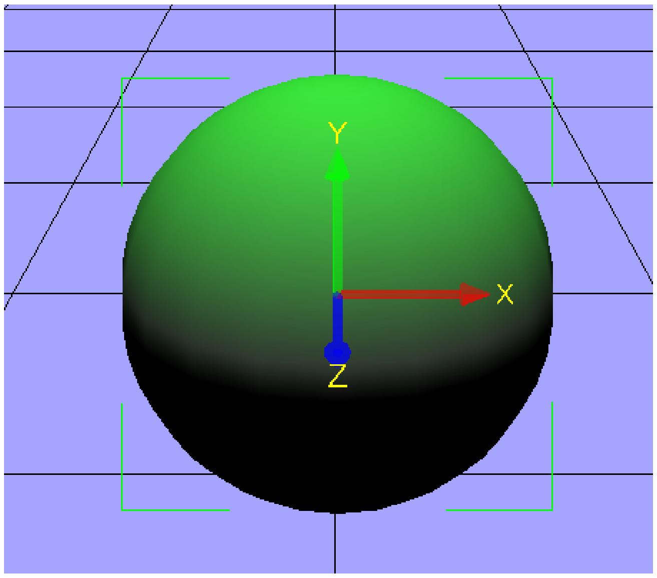
The Transform Manipulator allows the transformation of objects in the Scene Window. Picking on the axes of the manipulator allow constrained movement on the selected axis. Picking on the dual axis constraint plain confines movement to two axes.

Rotation Manipulator



The Rotation Manipulator allows the rotation of objects in the Scene Window. Picking on an arc constrain movement along that direction. Clicking on and dragging the gray sphere inside the arcs rotates the object in any direction.

Scale Manipulator

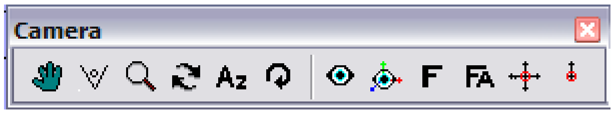


The Scale Manipulator allows the scaling of objects in the Scene Window. Scaling can only be done uniformly. Clicking and dragging left enlarges the object. Clicking and dragging right shrinks the object.

Toolbars

The Tables below describe the functionality of each Toolbar item.

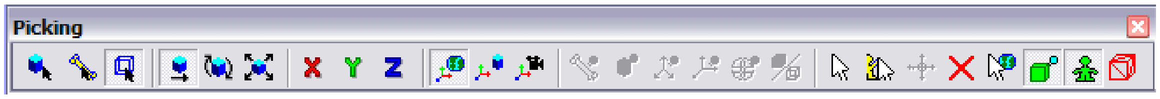
Camera Toolbar



Buttons are documented from left to right.

Translate Camera	Translates the active Camera on its X and Y axis.
Rotate Camera	Rotates the active Camera about its X and Y axis.
Zoom Camera	Translates the active Camera on it Z axis.
Roll Camera	Rotates the active Camera about its Z Axis.
Azimuth Camera	Changes the Azimuth and Elevation of the Camera in relation to the currently selected actor.
Tumble Camera	Rotates the Camera around the currently selected actor.
Look at Selected	Rotates the Camera so that it is looking at the currently selected actor.
Look at Origin	Rotates the Camera so that it is looking at the origin of the scene.
Frame Selected	Rotates and Translates the Camera so that the currently selected actor fills the view.
Frame All	Rotates and Translates the Camera so that all actors fill the view.
Reset Camera	Translates and Rotates the Camera to the default position for all Cameras. X=0,Y=16,Z=40,Xa=-0.5,Ya=0,Za=0
Upright Camera	Rotates the Camera so that its Y axis aligns with the Scene's Y axis.

Picking Toolbar



Buttons are documented from left to right.

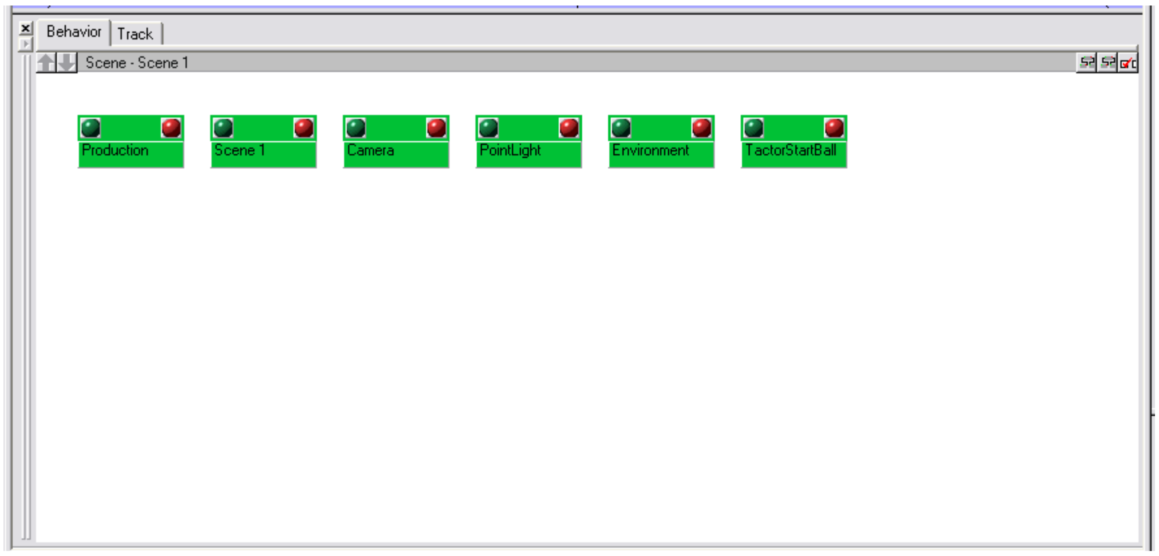
Pick Joints	Allows the individual joints of an actor to be picked.
Pick Bones	Allows the bones of an actor to be picked.
Pick Wire Frame	Allows objects displayed as wire frames to be picked.
Translate	Allows an actor or its joints to be translated
Rotate	Allows an actor or its joints to be rotated.
Scale	Allows an actor or its joints to be scaled.
Constrain X	Constrains transformations to the X axis when depressed.
Constrain Y	Constrains transformations to the Y axis when depressed.
Constrain Z	Constrains transformations to the Z axis when depressed.
Constrain Scene Space	Applies transformations relative to the scene when depressed.
Constrain Object Space	Applies transformations relative to the currently selected object when depressed.
Constrain Camera Space	Applies transformations relative to the camera when depressed.
Show Bones	Displays the bones of the currently selected actor when depressed.
Show Joints	Displays the joints of the currently selected actor when depressed.
Show Axes	Displays axes for the joints of the currently selected actor when depressed.
Show Transform Manipulator	Displays the transform manipulator for the currently selected actor when depressed.
Show Rotation Manipulator	Displays the rotation manipulator for the currently selected actor when depressed.
Show Joint Wireframes	Displays the currently selected joint in wire frame.
Pick Actor	Enables picking on the currently selected actor.
Pose Actor	Enables posing on the currently selected actor.

Reset Actor	Resets the currently selected actor to its original position.
Delete Actor	Deletes the currently selected actor from the scene.
Pick all Actors.	Enables picking on all actors in the scene.
Actor Selection Reticule	A green reticule is displayed around the currently selected joint or actor when depressed.
Actor Selection Mode	Allows the user to pick the currently selected actor by single clicking in the Scene Window.
Show Collision Volumes	Displays the bounding volume of an actor as wireframe when a collision event is active.

Accelerators

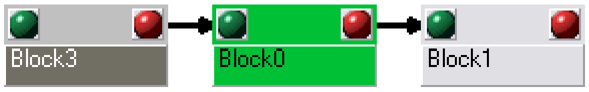
Undo Pick – Ctrl-Z	Undoes the last pick operation.

Behavior Window



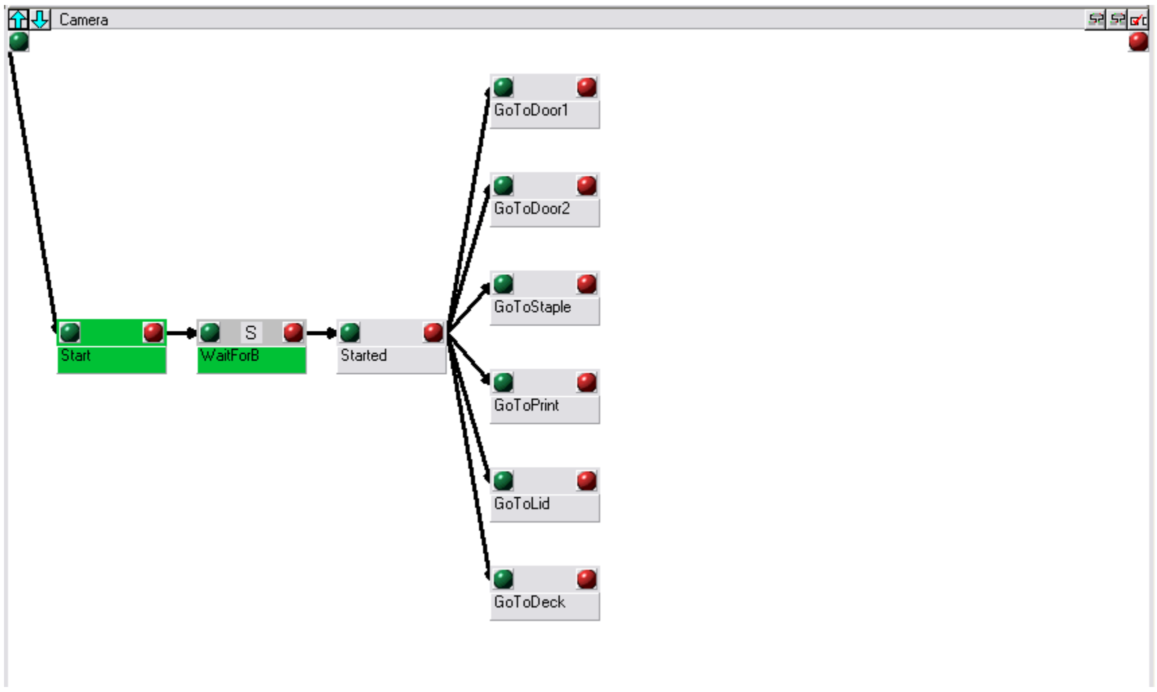
The Behavior Window is used for defining the behavior of entities. Its user interface provides a convenient method for assembling hierarchical finite state machines, the means by which behaviors are represented in the system. An entity's hierarchical finite state machine is symbolized as a directed cyclic graph similar to a Pert Chart. Individual behaviors are represented as blocks in the graph. Each entity in a production is represented by a top level behavior block called the meta-behavior. Within each meta-behavior a user creates a hierarchical directed graph of behaviors that define how the entity will act in certain situations.

Throughout their lifetime, behaviors cycle through three states: Idle, Active, and Done. The state of a behavior is denoted by changing its color in the view. Idle behaviors are colored gray, active behaviors are green, and done behaviors are dark gray.



These state changes are triggered by events which are boolean tests of the state of the production. Each behavior has a collection of Start and End events. When a behavior's start event evaluates to true, it transitions to the active state. When a behavior's end event is triggered the behavior enters the done state. The standard set of primitive event types are: Behavior Status, Keyboard, Collision, Scripted, Timer, Mouse, Ray Cast, Boolean, Animation, Joystick, Event Status, Relational, and Boolean. Links connecting pairs of behavior blocks represent behavior dependencies and implicitly the events that cause one behavior to terminate and another to begin. Links between behaviors are symbolized as lines that enter the behavior from the left and exit from the right. Behaviors are connected to one another by dragging a link from the end event of one behavior to the start event of another. To complete a linking operation the user is required to select an existing or configure a new start event to trigger the transition. By default a Behavior Status Event is placed on the originating behavior to terminate it when the start event of the destination behavior evaluates to true. Each event type has its own property page that allows the user to configure it. Additionally, primitive events can be combined into complex expressions for more involved evaluations of state.

Behaviors can also have sub-behaviors. A sub behavior can only be active if its parent behavior is active. Thus sub-behaviors can become active only when both the container and its own starting events evaluate to true. Stopping a parent behavior also stops its children. If a behavior contains sub-behaviors the behavior will have an icon to denote it. Sub-behaviors are accessed by double clicking on their container behavior. For a sub-behavior to be tested it must be linked to its container. The start and end event buttons in the upper left and upper right corners of the Behavior Window are the containers link points.



Behavior graphs for complicated productions can become quite large. To manage large graphs the Behavior Window provides controls for manipulating the view. The view can be zoomed in and out by holding the shift key and using the mouse's scroll wheel. Additionally, the view can be panned by either manipulating the scroll bars on the edge of the view or by holding down the shift key and moving the mouse with its left button depressed. Behaviors can be arranged in the view to the users liking by dragging them about the view or the user can use one of the auto arrange buttons on the view to sort the view according to a predefined algorithm. Mousing over a behavior block displays a tooltip with its name and type. Mousing over a behavior's start or end event buttons displays a tool tip containing the current list of events for that behavior. Behaviors can also be grouped, cut, copied and pasted to aid in authoring.

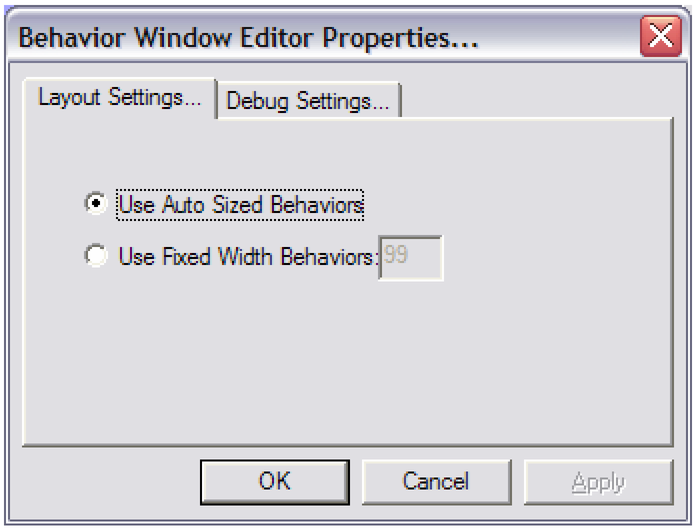
Buttons

<u>Behavior Window Buttons</u>

Buttons are documented left to right.

Navigate Up	Navigates the view one level up the hierarchy.
Navigate Down	Navigates the view one level down on the most recently selected block.
Arrange Blocks Left to Right	Arranges the block from left to right.
Arrange Blocks Top to Bottom	Arranges the block from top to bottom.
Behavior Window Properties	Opens the Behavior Window Properties Dialog from which the layout and debug setting for the view can be manipulated.

Behavior Window Properties Dialog



Use Auto Sized Behaviors [✓]
Use Fixed Width Behaviors: 99

The Behavior Window Properties dialog allows the user to configure the layout settings of the Behavior Window. Behaviors can be displayed so that their full name is visible regardless of the length of the behaviors name using the Auto Sized Behaviors option or they can specify a fixed display size for behaviors using the Fixed Width Behaviors option.

Menus

<u>Right Click Behavior View Menu</u>

Add Behavior	Adds an empty behavior to the view.
Restore	Navigates the view one level up the hierarchy.
Cut	Removes the currently selected behavior(s) from the view and places it in the clipboard.
Copy	Places a copy of the currently selected behavior(s) in the clipboard.
Paste	Places a copy of the clipboard contents within the current view.
Delete	Removes the currently selected behavior(s) from the view.

Export	Exports the currently selected behavior(s) to a file.
Import	Imports a behavior from a file.
Undo Layout	Nullifies the last layout operation.

Right Click Link Menu

<Start Event>	Opens the link properties dialog from which the user can modify the start and end events for the link that references <Start Event>
Delete All	Deletes all links
Delete <Start Event>	Deletes link that references <Start Event>.

Right Click Behavior Menu

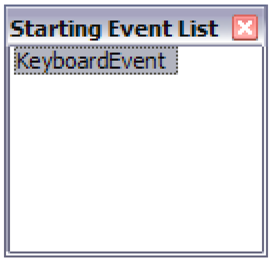
Add Parameter	Opens the Add Parameter Dialog.
Locate	Scrolls the Production Window to the currently selected behavior.
Run	Sets the currently selected behavior to the active state.
Stop	Stops the currently selected behavior and all its children.
Stop Children	Stops the currently selected behavior's children.
Cut	Removes the currently selected behavior from the view and places it in the clipboard.
Copy	Places a copy of the currently selected behavior in the clipboard.
Paste	Places a copy of the clipboard contents within the current view.

Delete	Removes the current behavior from the view.
Find	Opens the Find dialog.
Export	Exports the currently selected behavior to a file.
Import	Imports a behavior from a file.
Group	Creates a Container behavior and makes the current selection it Sub-behavior(s).
Pin Down	Prevents the currently selected behavior from being moved in the view.

Mouse Operations

Shift Left Click	Pans the view.
Shift Scroll	Zooms the view.

Start/End Event List Popup



Pop up window that shows the collection of start/end events attached to this behavior. The window is accessed by clicking on either the green button (start events) or red button (end events) located on the left and right side of the behavior respectively. Mousing over a event displays a tool tip containing that event's event expression.

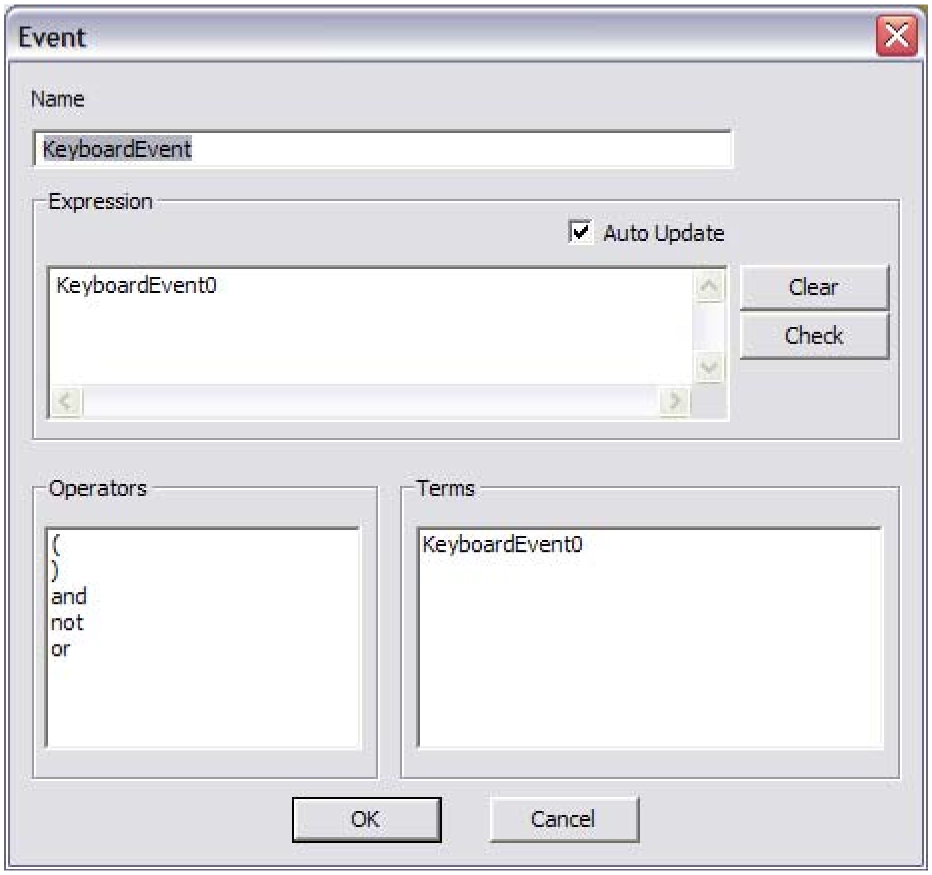
Menus

Right Click Event List Item Menu

Edit	Opens the Event Expression dialog for the currently selected event.
Cut	Removes the currently selected event from the list and places it in the clipboard.
Copy	Places a copy of the currently selected event in the clipboard.
Paste	Places a copy of the clipboard contents within the current list.
Delete	Removes the currently selected event from the list.
Export	Exports the currently selected event to a file.
Import	Imports an event from a file.
New\Behavior Status	Opens the Behavior Status Event dialog.
New\Keyboard	Opens the Keyboard Event dialog.
New\Collision	Opens the Collision Event dialog.
New\Scripted	Opens the Script Event dialog.
New\Timer	Opens the Timer Event dialog.
New\Mouse	Opens the Mouse Event dialog.
New\Ray Cast	Opens the Ray Cast Event dialog.
New\Animation	Opens the Animation Event dialog.
New\Joystick	Opens the Joystick Event dialog.
New\Event Status	Opens the Event Status Event dialog.
New\Boolean	Opens the Boolean Event dialog.

New\Relational	Opens the Relational Event dialog.
New\Complex	Opens the Complex Event dialog.

Event Dialog



The Event dialog allows the user to create complex event expressions. Event expressions are composed of primitive events that act as terms and operators that evaluate those terms to produce a boolean result. The result of an event expression is used to trigger events. Primitive events can be created and used as terms in the Terms List Box and then combined with operators from the Operators List Box to produce an expression in the expression editor. Terms and operators can be added to the expression editor by double clicking them in their list boxes. Right clicking the Terms List Box provides a menu of operations for managing the set of available terms such as copy, paste, import, export, etc. If the user prefers they may type the event expression directly into the expression text box. Right clicking in the expression edit box provides familiar text editing operations. Additionally, a check button is provided to verify that the expression is valid. When the Auto Update option is checked the expression will be updated when a term is renamed or deleted.

Menus

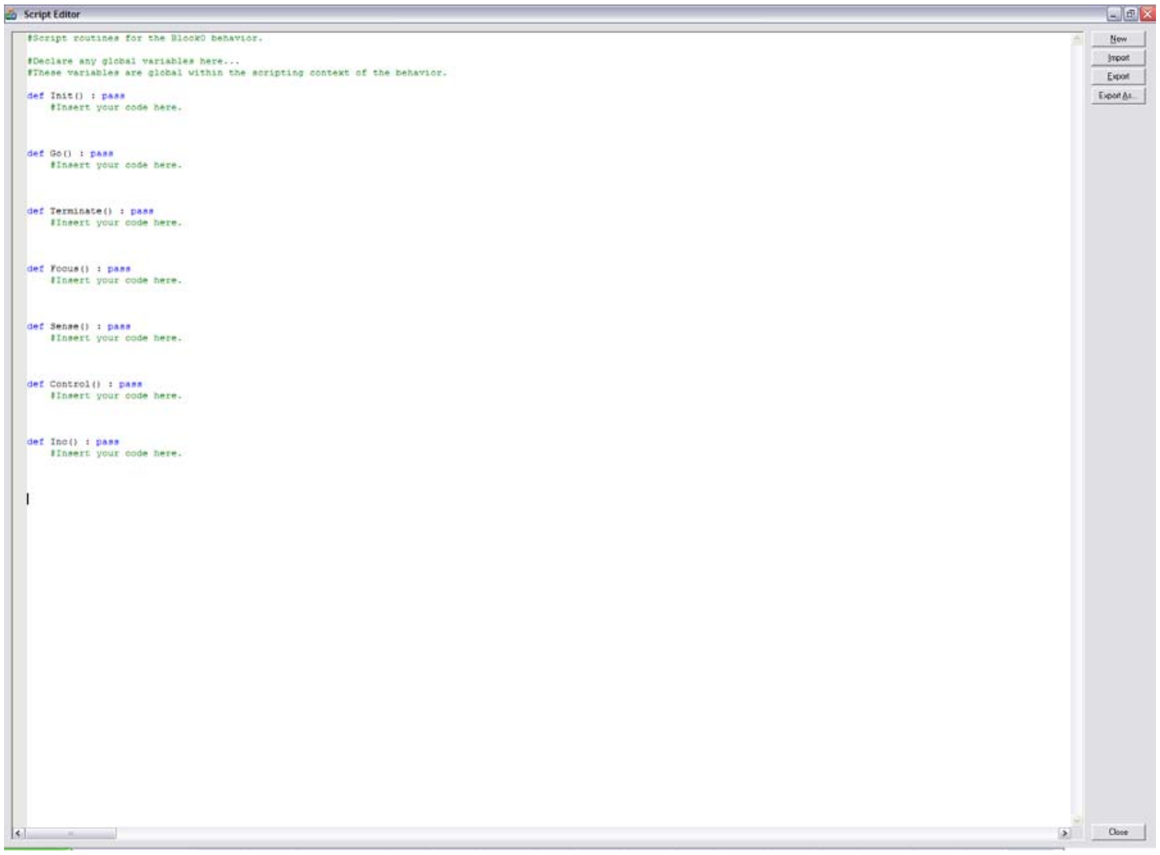
Expression Text Box Right Click Menu

Undo	Nullifies the last operation.
Cut	Removes the currently selected text from the text box and places it in the clipboard.
Copy	Places a copy of the currently selected text in the clipboard.
Paste	Places a copy of the clipboard contents within the current list.
Delete	Removes the currently selected text from the text box.
Select All	Selects all of the text in the text box.

Terms List Box Right Click Menu

Edit	Opens the currently selected term’s event dialog.
Cut	Removes the currently selected term from the list box and places it in the clipboard.
Copy	Places a copy of the currently selected term in the clipboard.
Paste	Places a copy of the clipboard contents within the current list.
Delete	Removes the currently selected term from the list box.
Export	Exports the currently selected term to a file.
Import	Imports a term from a file.
New\Behavior Status	Opens the Behavior Status Event dialog.
New\Keyboard	Opens the Keyboard Event dialog.
New\Collision	Opens the Collision Event dialog.
New\Scripted	Opens the Script Event dialog.
New\Timer	Opens the Timer Event dialog.
New\Mouse	Opens the Mouse Event dialog.
New\Ray Cast	Opens the Ray Cast Event dialog.
New\Animation	Opens the Animation Event dialog.
New\Joystick	Opens the Joystick Event dialog.
New\Event Status	Opens the Event Status Event dialog.
New\Boolean	Opens the Boolean Event dialog.
New\Relational	Opens the Relational Event dialog.
New\Complex	Opens the Complex Event dialog.

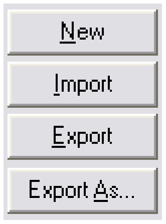
Script Editor



```
#Script routines for the Block0 behavior.

#Declare any global variables here...
#These variables are global within the scripting context of the behavior.

def Init() : pass
    #Insert your code here.


def Go() : pass
    #Insert your code here.


def Terminate() : pass
    #Insert your code here.


def Focus() : pass
    #Insert your code here.


def Sense() : pass
    #Insert your code here.


def Control() : pass
    #Insert your code here.


def Inc() : pass
    #Insert your code here.
```

The Script Editor is a word processing utility for displaying and editing script code. Each script is represented as a tab. Language Profiles can be used to color code the text of the script according to the scripting language in use. Scripts are stored in a Production but can also be exported and imported to the file system for reuse. The right hand side of the Script Window has buttons for managing scripts. Right clicking in the text editor reveals the available options for authoring code. Additionally, keyboard accelerators are available for launching the search and replace functions of the editor.

Buttons



The Table below describes the functionality of each script management button.

New	Creates a new script.
Import	Opens an existing script.
Export	Saves the script to the file system.
Export As	Saves a script to the file system at a user defined location.
Close	Close the script error. If modifications have been made to the script a dialog will prompt the user to save the changes.

Menus

Text Editor Right Click Menu

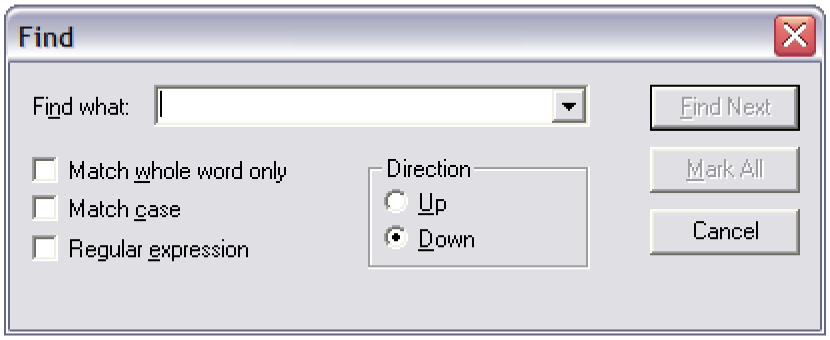
The Table below describes the functionality of each menu item.

Undo	Nullifies the last action.
Redo	Performs the last action you undid.
Cut	Removes the currently selected term from the list box and places it in the clipboard.
Copy	Places a copy of the currently selected term in the clipboard.
Paste	Places a copy of the clipboard contents within the current list.
Select All	Selects all text within the script.

Text Editor Accelerators

Ctrl-F	Opens the Find dialog.

Find Dialog


Find

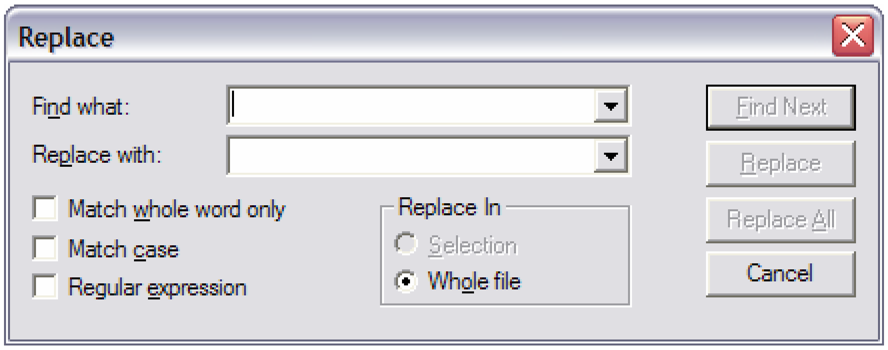
Find what:

☐ Match whole word only
☐ Match case
☐ Regular expression

Direction
○ Up
● Down

The Find Dialog allows for text search of the script. One can do simple match searches or use regular expressions for more advanced searching. The dialog supports perl regular expression syntax.

Replace Dialog


Replace

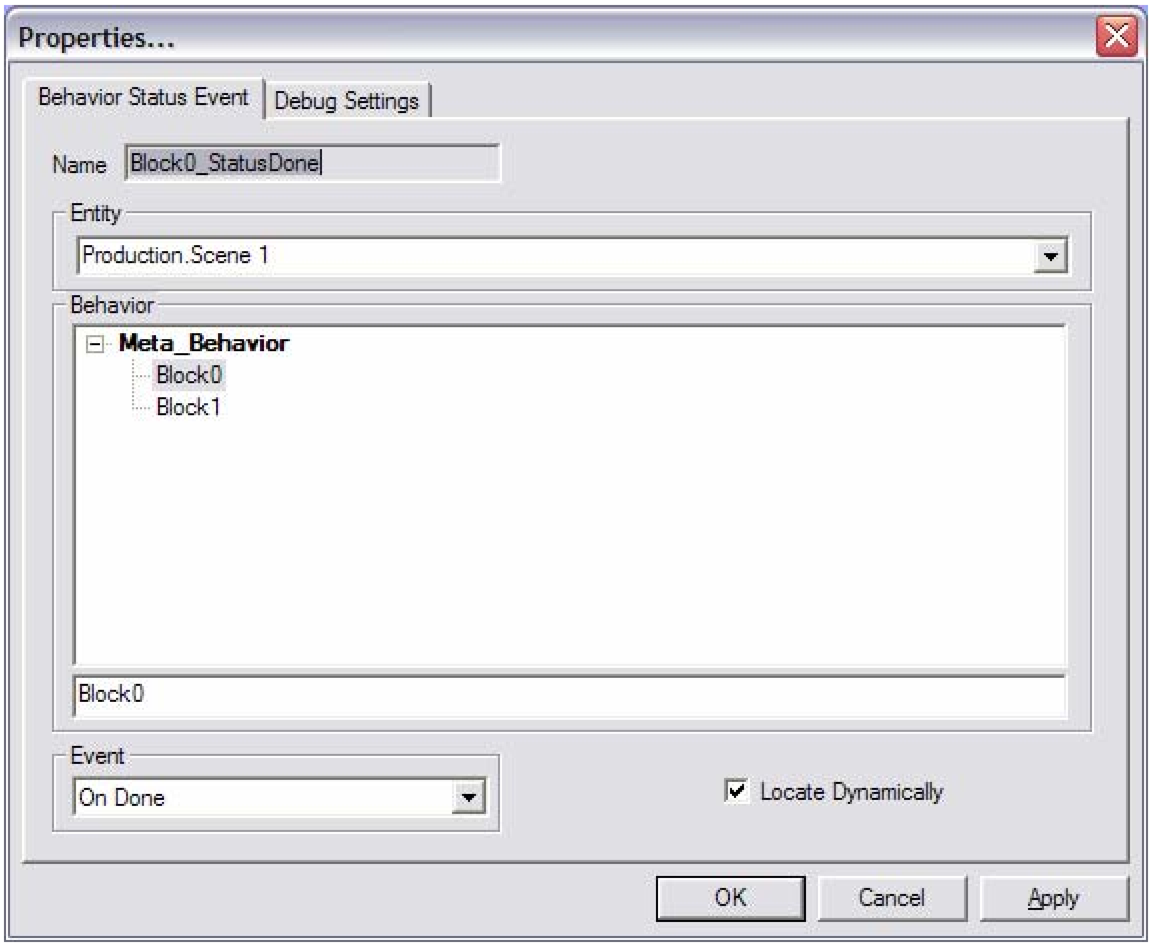
Find what:
Replace with:

☐ Match whole word only
☐ Match case
☐ Regular expression

Replace In
○ Selection
● Whole file

The Find Dialog allows for text replace in the document. One can do simple match replaces or use regular expressions for more advanced replace operations. The dialog supports perl regular expression syntax.

Behavior Status Event Dialog


Properties...

Behavior Status Event | Debug Settings

Name: Block0_StatusDone

Entity: Production.Scene 1

Behavior:
- Meta_Behavior
  - Block0
  - Block1

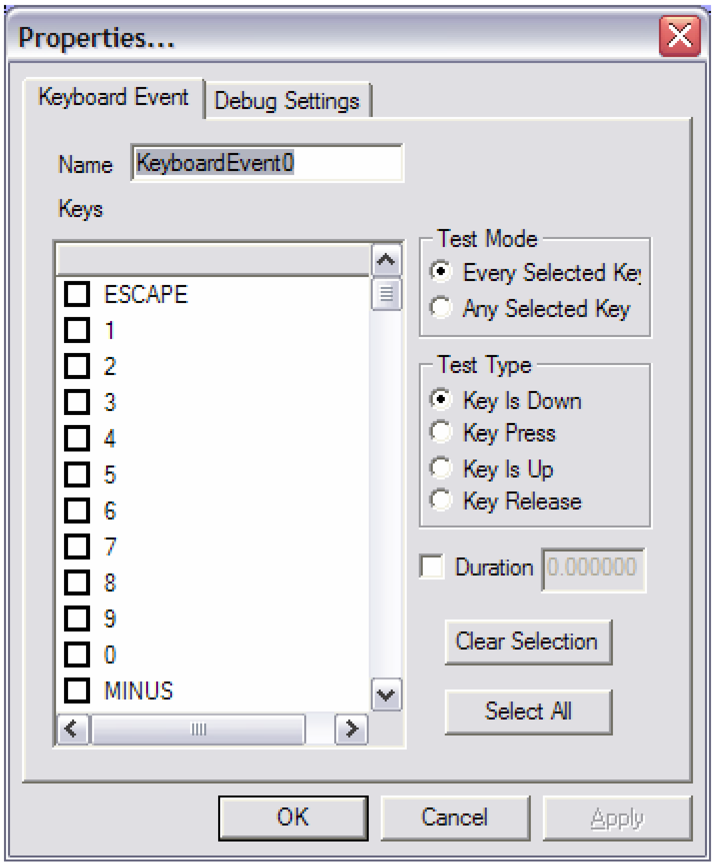
Block0

Event: On Done

[✓] Locate Dynamically

The Behavior Status Event Dialog allows the user to specify events which monitor the state of behaviors in a production. The Entity Combo Box provides a drop down list of entities in the production to choose from. Additionally, the user can type in the name of an entity in anticipation of its creation. The dialog expects a name using dotted notation of the following form <Production>.<Scene>.<Actor>. Once an entity is selected the Behavior tree displays a list of its available behaviors to choose from. Optionally, the user can enter the name of a behavior in anticipation of its creation. If the Production's Full Encapsulation property is set to true only those Entities that are in scope will be capable of being referenced. With the entity and behavior specified the user can select a status event to monitor. The set of observable events include idle, active, done, pending done, sub-behavior start, sub-behavior end, sub-behavior active, sub-behavior inactive, and exists.

Keyboard Event Dialog


Properties...

Keyboard Event | Debug Settings

Name: KeyboardEvent0

Keys
- [ ] ESCAPE
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7
- [ ] 8
- [ ] 9
- [ ] 0
- [ ] MINUS

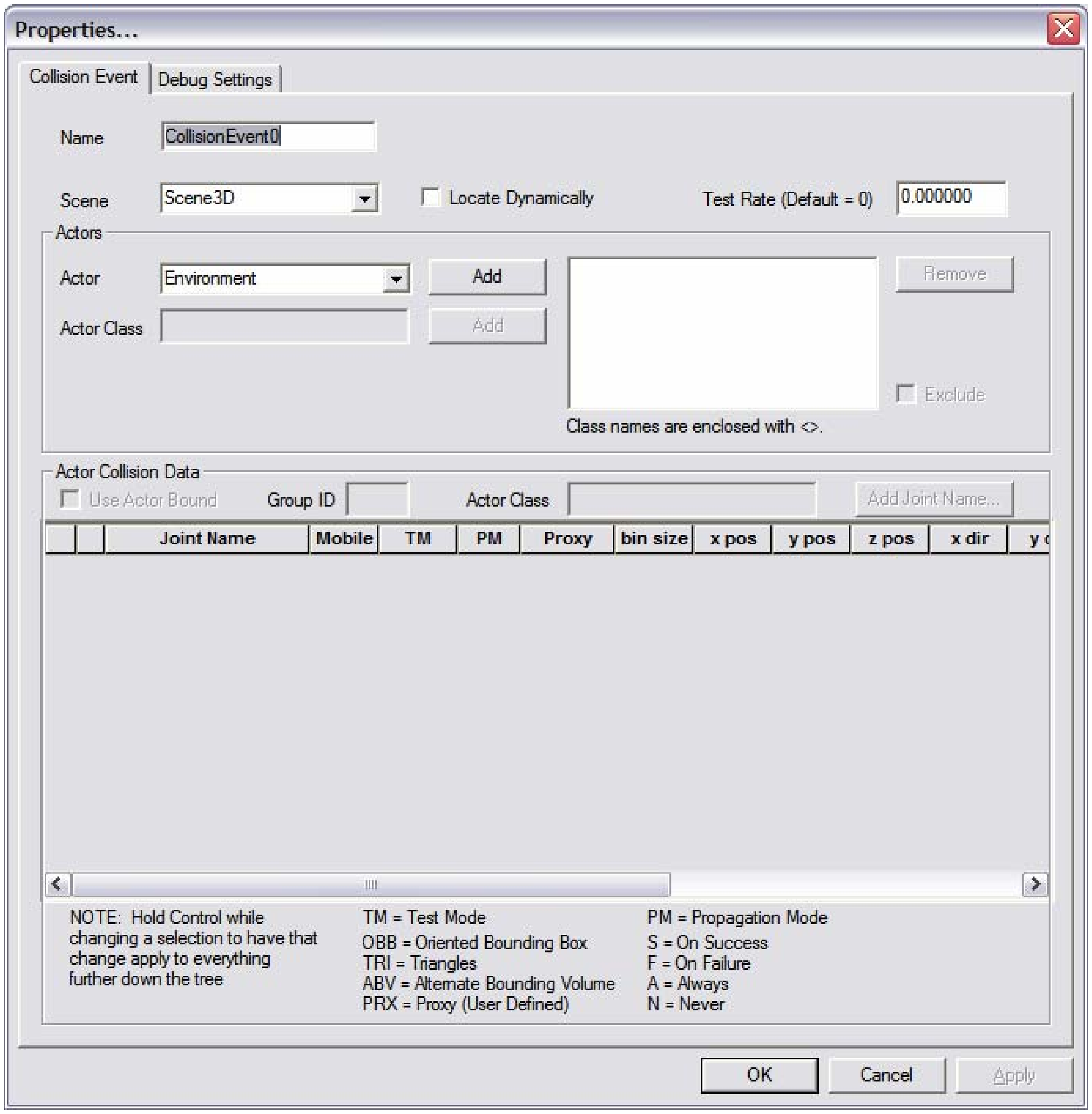
Test Mode: [● Every Selected Key] ○ Any Selected Key

Test Type: [● Key Is Down] ○ Key Press ○ Key Is Up ○ Key Release

- [ ] Duration 0.000000

The Keyboard Event Dialog allows the user to specify events which monitor the state of the keyboard. The user can specify keys or chords to watch for an up, down, press and release states. Additionally, the user can also specify a duration for which the keys must be in one of these four states in order for the event to trigger.

Collision Event Dialog



Name: CollisionEvent0

Scene: Scene3D

Locate Dynamically

Test Rate (Default = 0): 0.000000

Actors

Actor: Environment

Actor Class

Class names are enclosed with <>.

Actor Collision Data

Use Actor Bound

Group ID

Actor Class

		Joint Name	Mobile	TM	PM	Proxy	bin size	x pos	y pos	z pos	x dir	y c

NOTE: Hold Control while changing a selection to have that change apply to everything further down the tree

TM = Test Mode
OBB = Oriented Bounding Box
TRI = Triangles
ABV = Alternate Bounding Volume
PRX = Proxy (User Defined)

PM = Propagation Mode
S = On Success
F = On Failure
A = Always
N = Never

The Collision Event allows users to monitor the intersection of two or more actors and/or their associated bounding volumes. To specify a collision event one must specify a set of actors to test for collisions. This is accomplished by populating the Actors List. The Scene and Actor Combo Boxes allow you to choose an actor to test. Once an actor is chosen add the actor to the Actor List by pressing the Add Button. To remove an actor on the Actors List press the Remove Button.

Once the subjects of the collision have been chosen the parameters of the collision test must be specified. The Collision List displays a hierarchical list of joints and the associated parameters for the collision test you wish to perform on the currently selected actor in the Actors List. The user must select each actor in turn from the Actor List to configure its collision test parameters in the Collision List.

One configures a collision test by specifying parameters for each joint that should be tested. Each available joint has its name listed in the Joint Name column. The far left column contains check boxes that allow the user to specify which joints to include in the test. Joints are listed in a hierarchy mirroring the joint structure of the actor. Holding the Ctrl key while clicking will include all sub-joints in a selection as well. Each column contains a data entry field that specifies a parameter for the test. At first a limited set of options is available for configuration. As the user narrows down their initial choices more or less options are made available for configuration. The following table describes the set of available options.

Mobile	Specifies that the object will move during testing. Objects can be denoted as mobile are tested more rigorously than those that are not.
TM	Specifies the Test Mode for the test. The available options are: OBB – Oriented Bounding Boxes. Indicates that collision is to be done by creating an efficient oriented bounding box for the associated geometric data. Choosing this option will make the Bin Size column available for configuration. Tri – Triangles. Indicates to collision that the object wishes to use it's raw triangles for collision detection ABV – Alternate Bounding Volume. Indicates that alternate bounding volumes are to be used. This flag should only be used if an ABV has actually been constructed for the collision data. When an actor using this flag encounters another using ABVs, the most optimal collision detection can occur. If no ABV is provided on the model an ABV will be automatically generated based on the selection in the Proxy Column. PRX – Proxy. Indicates that the user will specify the bounding volume in the Proxy Column.
PM	Specifies the propagation mode for the test. The propagation parameter give the user a mechanism to precisely direct operations of the collision detection system as it searches for intersections. The available options are: S – On Success. Indicates that the propagation down the scene graph should occur upon successful

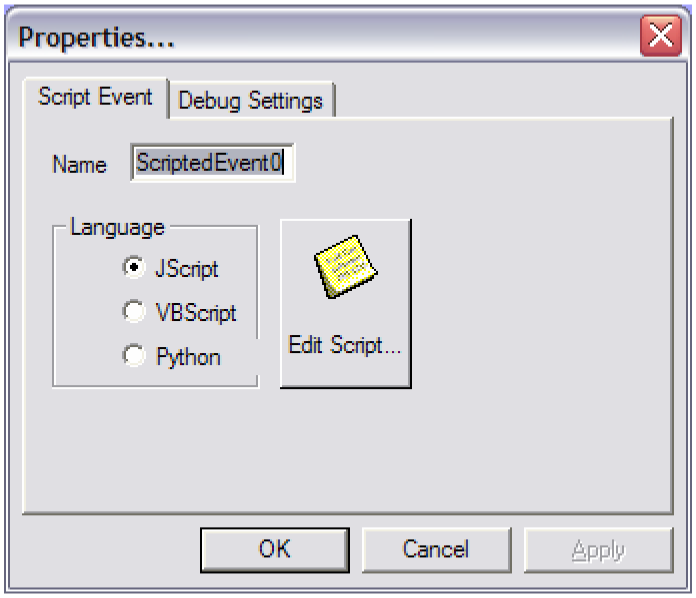
	collision detection. F – On Failure. Indicates that the propagation down the scene graph should only occur upon a failure of collision detection. A – Always. Indicates that propagation down the scene graph should occur whether there was a collision or not. N – Never. Indicates that regardless of the collision results, propagation is never to occur.
Proxy	A list of bounding volumes that can be used if no ABV is specified or the user has elected to specify his own proxy for the test. The available options are: Sphere, Box, Capsule, Half Space. The choice made here will allow the configuration of more options in columns to the right of this one. These additional options all the user to specify the parameters for the bounding volume.
Bin Size	Available only when OBB is specified for the Test Mode. Bin Size indicates the number of triangles per leaf node in the OBB tree.
XPos, YPos, ZPos	Available only when Proxy is specified for the Test Mode. These parameters specify the X, Y, and Z location of the proxy in local coordinates.
XDir, YDir, ZDir	Available only when Proxy is specified for the Test Mode and the Proxy is either Capsule or Half Space. Specifies the X, Y, and Z axis that the Length parameter refers to for Capsule. Specifies the direction of the normal for Half Space.
XSize, YSize, ZSize	Available only when Proxy is specified for the Test Mode and the Proxy is Box. Specifies the X, Y, and Z lengths of edges of the box.
Radius	Available only when Proxy is specified for the Test Mode and the Proxy is Sphere or Capsule. Specifies the radius of the proxy.
Length	Available only when Proxy is specified for the Test Mode and the Proxy is Capsule. Specifies the length of the Capsule.

The rate at which collision are tested is specified in the Collision Test Rate edit box. The default rate is zero which means tests are performed every frame. Specifying a higher number increases the time between tests. For example specifying a rate of 10 means a collision test is performed every 10 frames.

One may also test whether the inverse of a collision test is true. By selecting the Not Colliding option the collision test will be true when the specified actors are not intersecting.

If one wishes to test collisions for an entire actor the Actor Bound option should be selected. The Actor Bound test calculates a single bounding volume that encompasses all of an actor's joints. Checking the Mobile option in the Collision List The bounding volume for the test can be specified in Test Mode column of the Collision List. Additionally, the user may exclude an actor from collision testing with certain other actors by including them in a group denoted by the Group Id edit box. Specifying the same number for a collection of actors excludes them from collision tests against each other.

Scripted Event Dialog

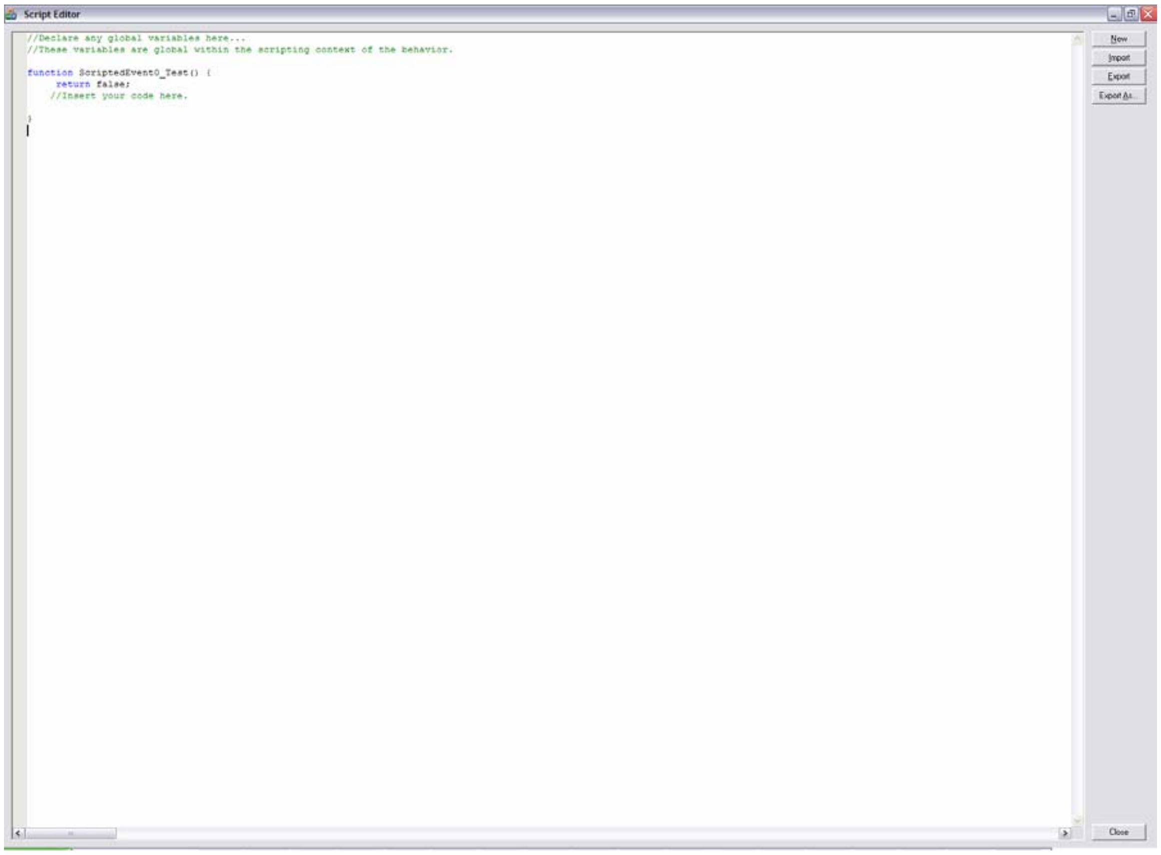


Name: ScriptedEvent0

Language: [✓ JScript] VBScript Python

The Script Event Dialog allows the user to write a Script in one of the system supported scripting languages. The user can choose the script language to use via the radio button. Pressing the edit script button launches the Script Editor. The script editor is filled in with a blank template in the chosen language for newly created script events. The user may change the language in use at any time.

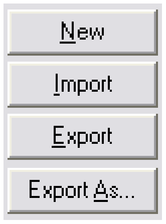
Script Event Editor



```
//Declare any global variables here...
//These variables are global within the scripting context of the behavior.

function ScriptedEvent0_Test() {
     return false;
    //Insert your code here.

}
```

The Script Editor is a word processing utility for displaying and editing script code. Each script is represented as a tab. Language Profiles can be used to color code the text of the script according to the scripting language in use. Scripts are stored in a Production but can also be exported and imported to the file system for reuse. The right hand side of the Script Window has buttons for managing scripts. Right clicking in the text editor reveals the available options for authoring code. Additionally, keyboard accelerators are available for launching the search and replace functions of the editor.

Buttons



The Table below describes the functionality of each script management button.

New	Creates a new script.
Import	Opens an existing script.
Export	Saves the script to the file system.
Export As	Saves a script to the file system at a user defined location.
Close	Close the script error. If modifications have been made to the script a dialog will prompt the user to save the changes.

Menus

Text Editor Right Click Menu

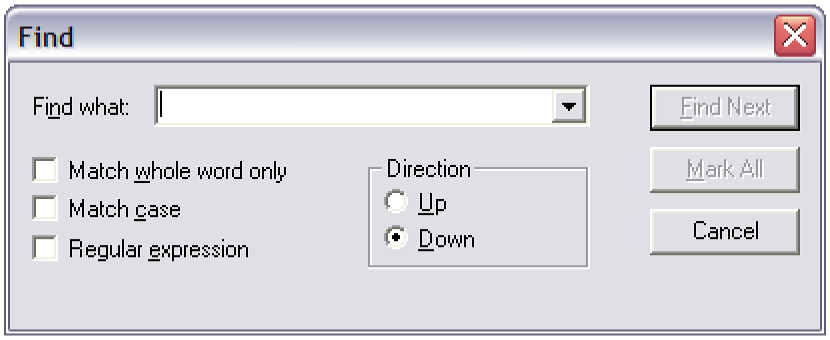
The Table below describes the functionality of each menu item.

Undo	Nullifies the last action.
Redo	Performs the last action you undid.
Cut	Removes the currently selected term from the list box and places it in the clipboard.
Copy	Places a copy of the currently selected term in the clipboard.
Paste	Places a copy of the clipboard contents within the current list.
Select All	Selects all text within the script.

Accelerators

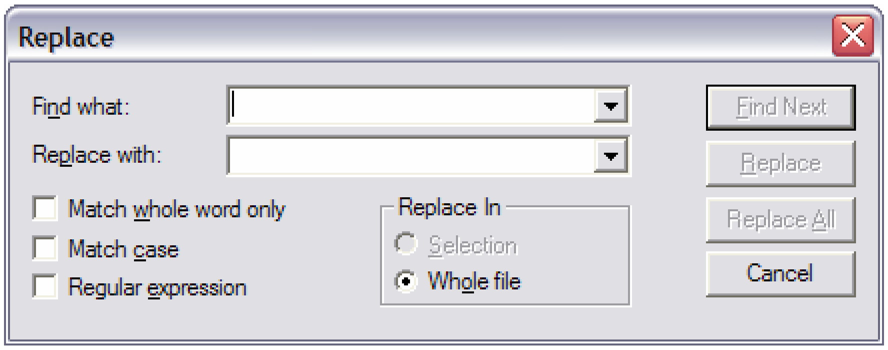
Ctrl-F	Opens the Find dialog.

Find Dialog



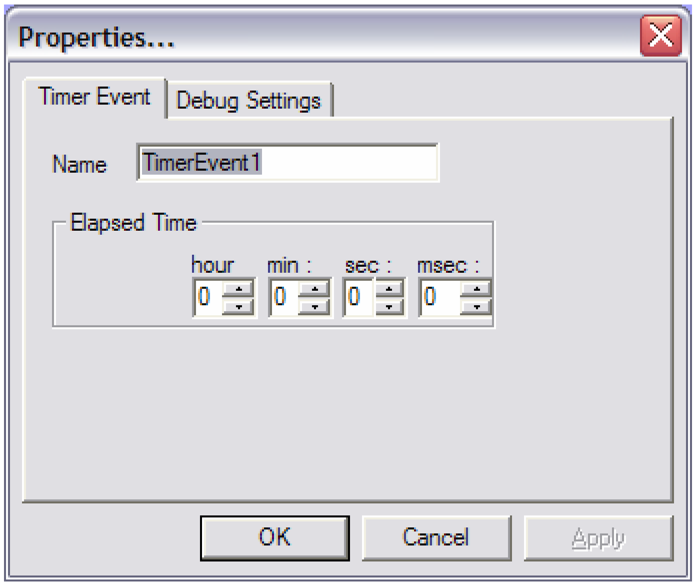
The Find Dialog allows for text search of the script. One can do simple match searches or use regular expressions for more advanced searching. The dialog supports perl regular expression syntax.

Replace Dialog



The Find Dialog allows for text replace in the document. One can do simple match replaces or use regular expressions for more advanced replace operations. The dialog supports perl regular expression syntax.

Timer Event Dialog



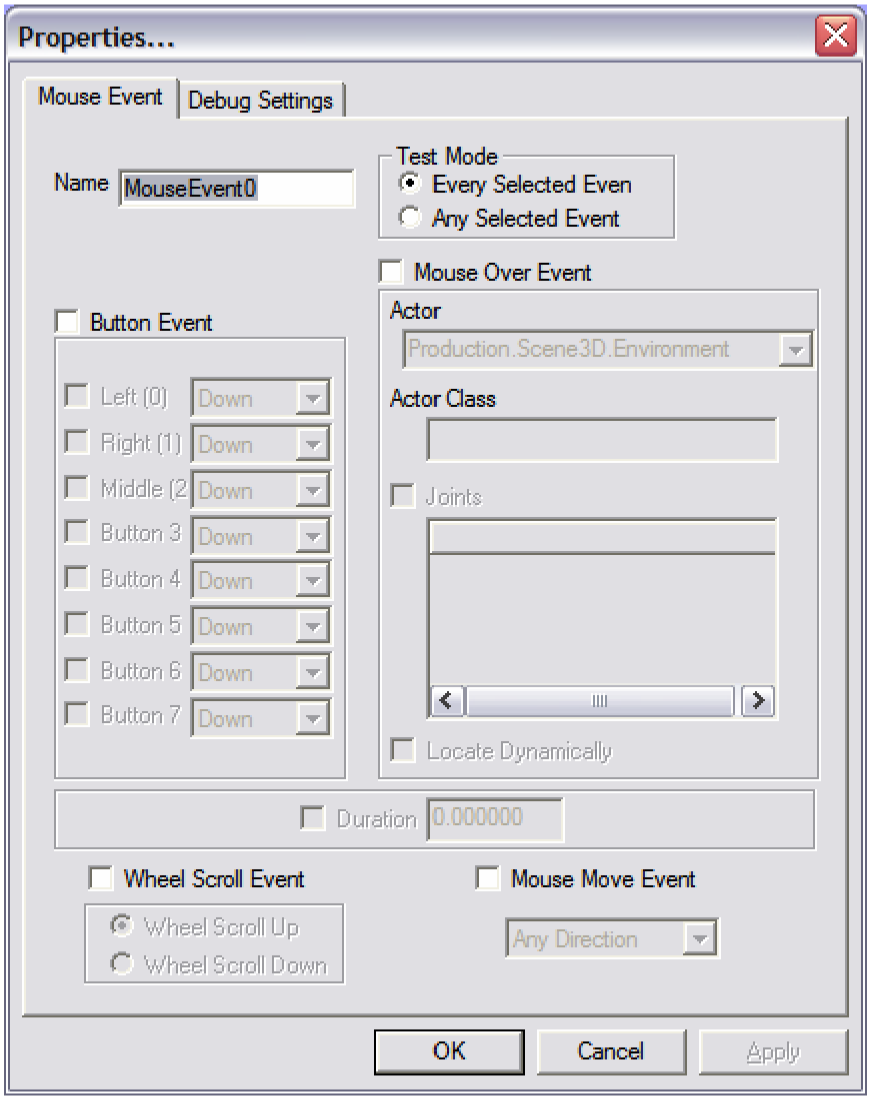
Name: TimerEvent1

Elapsed Time

hour: 0 min: 0 sec: 0 msec: 0

The Timer Event Dialog allows the user to specify a timer event. Timer events require the user to specify an activation time. The activation time is measured from the moment that the event begins being tested.

Mouse Event Dialog



Name: MouseEvent0

Test Mode: [✓ Every Selected Even] Any Selected Event

☐ Mouse Over Event

Actor: Production.Scene3D.Environment

Actor Class:

☐ Joints

☐ Locate Dynamically

☐ Button Event

☐ Left (0)	Down
☐ Right (1)	Down
☐ Middle (2	Down
☐ Button 3	Down
☐ Button 4	Down
☐ Button 5	Down
☐ Button 6	Down
☐ Button 7	Down

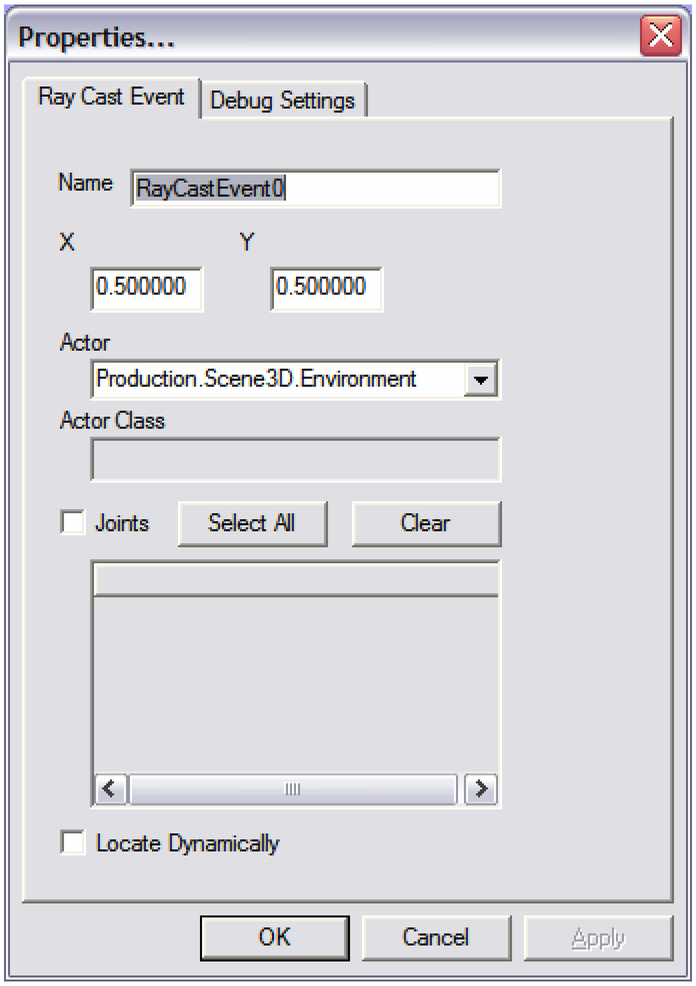
☐ Duration 0.000000

☐ Wheel Scroll Event: [✓ Wheel Scroll Up] Wheel Scroll Down

☐ Mouse Move Event: Any Direction

The Mouse Event Dialog allows the user to monitor mouse state. Mouse events can monitor any or all of the following states: button activity (i.e. down, up, release, press, and double click), cursor over an object, cursor movement direction, and scroll wheel movement direction. Additionally, the user can specify a duration threshold for the button activity and cursor activity that triggers the event.

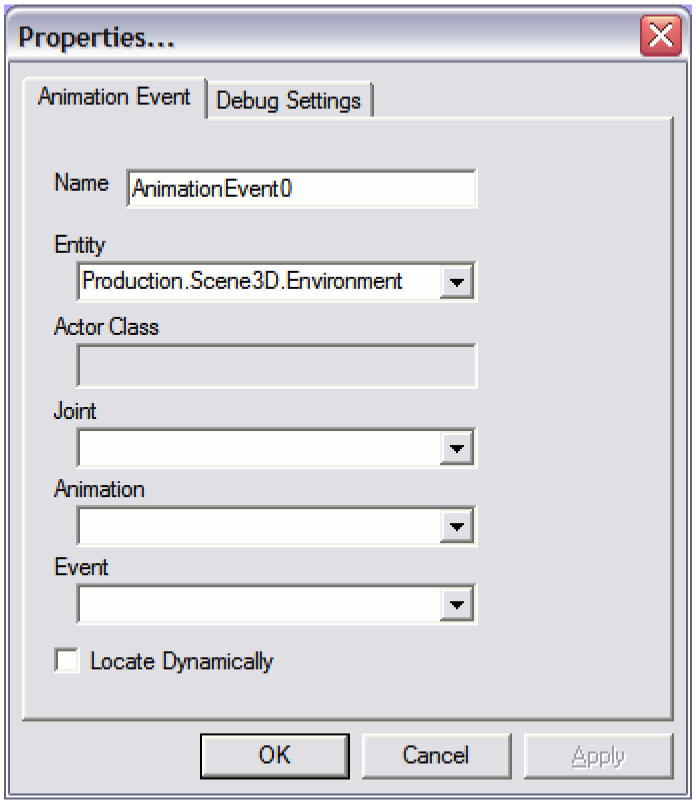
Ray Cast Event Dialog



Name: RayCastEvent0
X: 0.500000
Y: 0.500000
Actor: Production.Scene3D.Environment
Actor Class:
☐ Joints
☐ Locate Dynamically

The Ray Cast Event Dialog allows the user to specify a collision test by casting a ray into the scene from a designated point in screen space. The user specifies what screen coordinates to use to cast the ray and the actor's joints to monitor for the collision.

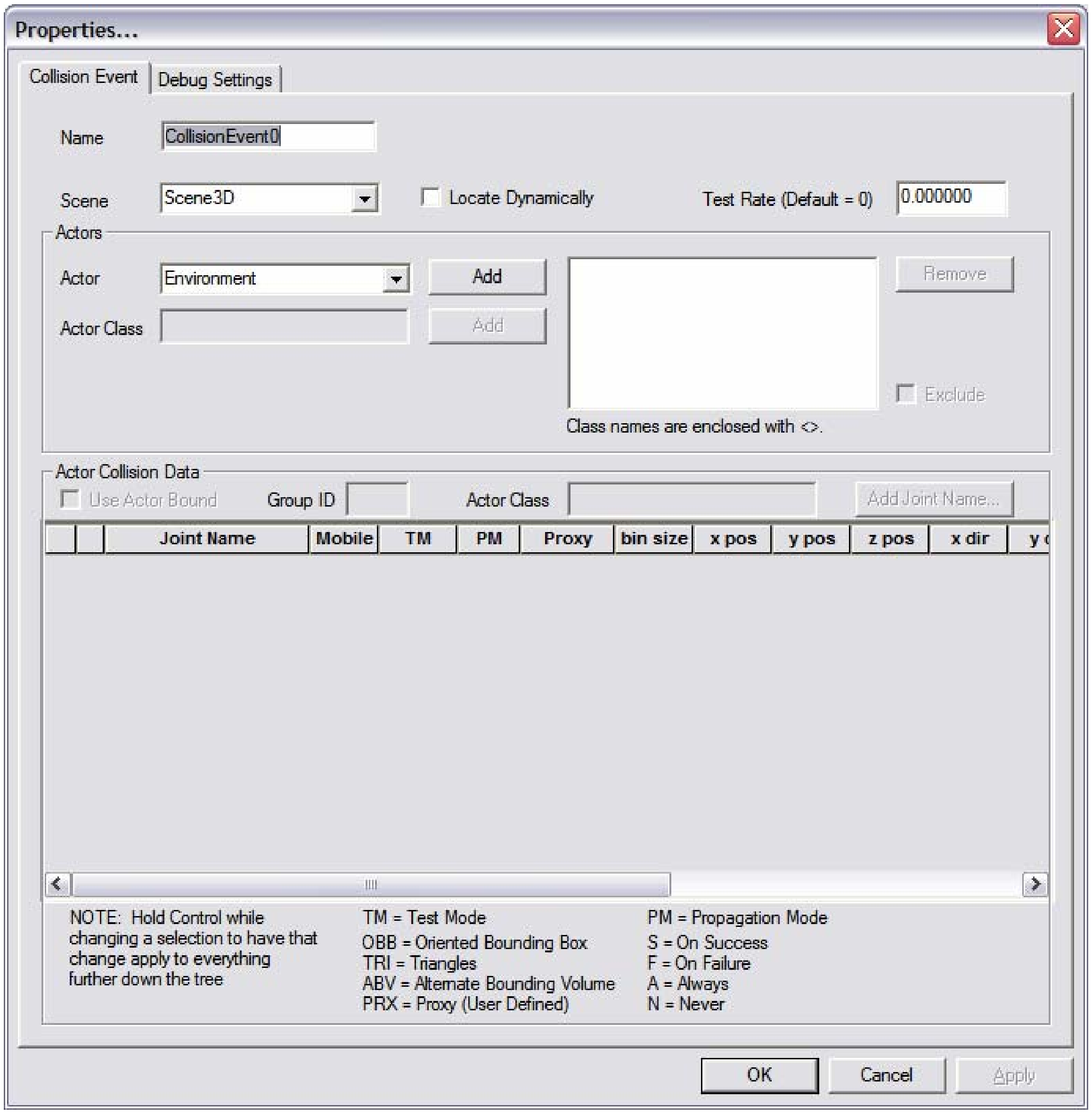
Animation Event Dialog



Name: AnimationEvent0
Entity: Production.Scene3D.Environment
Actor Class:
Joint:
Animation:
Event:
☐ Locate Dynamically

The Animation Event Dialog allows the user to monitor text events being issued from the animation system. Text events are placed on an actor's animations in the artist's tools; they designate important points in the animation which can be used to trigger other activity in the production. All animations have start and end events. Additional events can be added at the discretion of the artist.

Joystick Event Dialog



Properties...

Collision Event | Debug Settings

Name: CollisionEvent0

Scene: Scene3D

☐ Locate Dynamically

Test Rate (Default = 0): 0.000000

Actors

Actor: Environment — Add

Actor Class: — Add

Remove

☐ Exclude

Class names are enclosed with <>.

Actor Collision Data

☐ Use Actor Bound — Group ID — Actor Class — Add Joint Name...

		Joint Name	Mobile	TM	PM	Proxy	bin size	x pos	y pos	z pos	x dir	y d

NOTE: Hold Control while changing a selection to have that change apply to everything further down the tree

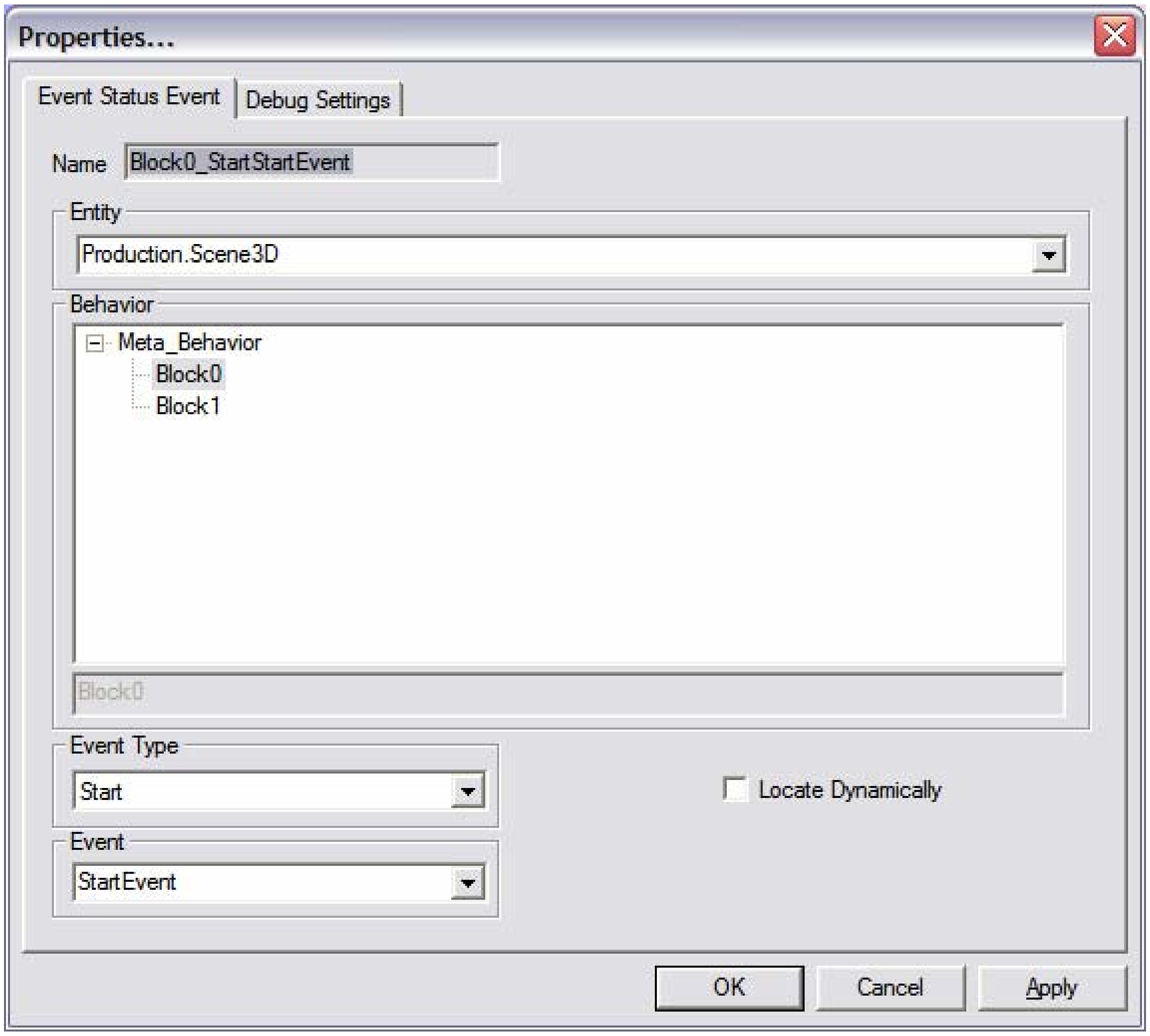
TM = Test Mode
OBB = Oriented Bounding Box
TRI = Triangles
ABV = Alternate Bounding Volume
PRX = Proxy (User Defined)

PM = Propagation Mode
S = On Success
F = On Failure
A = Always
N = Never

OK | Cancel | Apply

The Joystick Event Dialog allows the user to monitor joystick activity. The joystick event is capable of observing any or all Movement, Range, and Centering events for available axis/pov's as well as Up, Down, Press, and Release events for any present buttons. Duration can also be factored in to the event test.

Event Status Event Dialog



Name: Block0_StartStartEvent

Entity: Production.Scene3D

Behavior:
- Meta_Behavior
  - Block0
  - Block1

Block0

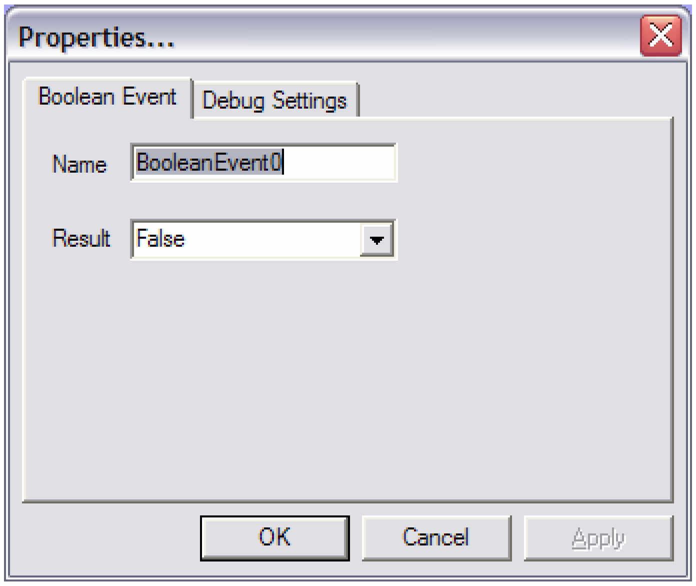
Event Type: Start

Event: StartEvent

☐ Locate Dynamically

The Event Status Event Dialog allows users the ability to monitor the status of other events in the production. Using the dialog consists of choosing an event to monitor by filtering the set of available events by first selecting the entity, behavior, and event type (i.e. start or end) .

Boolean Event Dialog

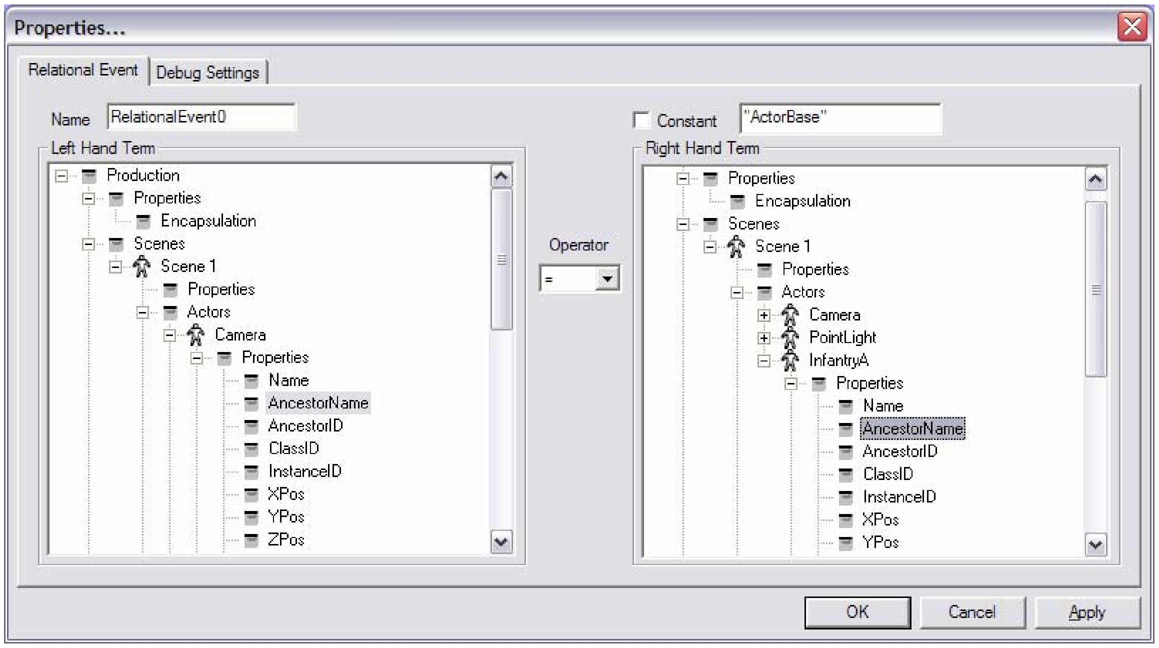


Name: BooleanEvent0

Result: False

The Boolean Event Dialog is used to configure events that trigger as always false or always true.

Relational Event Dialog



Name: RelationalEvent0

Constant: "ActorBase"

Operator: =

Left Hand Term:

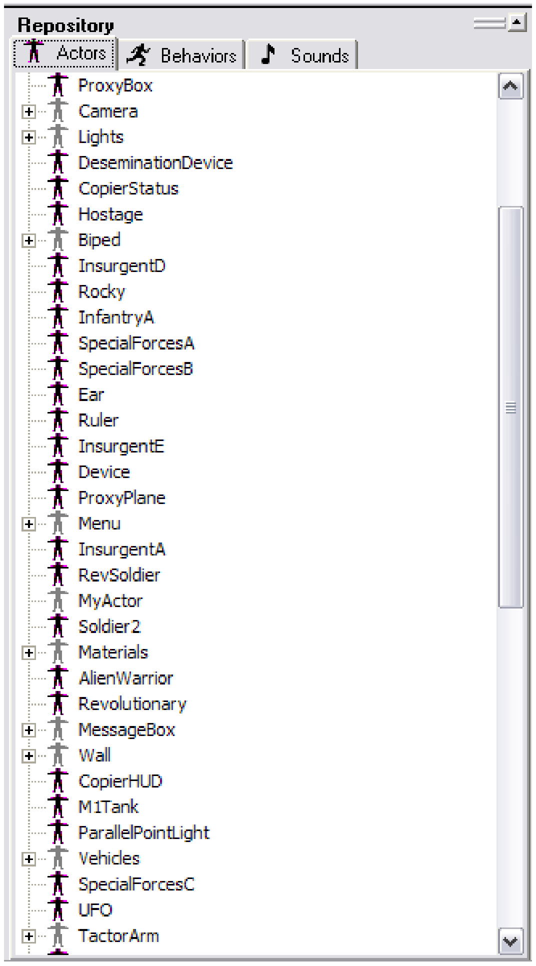
```
Production
  Properties
    Encapsulation
  Scenes
    Scene 1
      Properties
      Actors
        Camera
          Properties
            Name
            AncestorName
            AncestorID
            ClassID
            InstanceID
            XPos
            YPos
            ZPos
```

Right Hand Term:

```
Properties
  Encapsulation
Scenes
  Scene 1
    Properties
    Actors
      Camera
      PointLight
      InfantryA
        Properties
          Name
          AncestorName
          AncestorID
          ClassID
          InstanceID
          XPos
          YPos
```

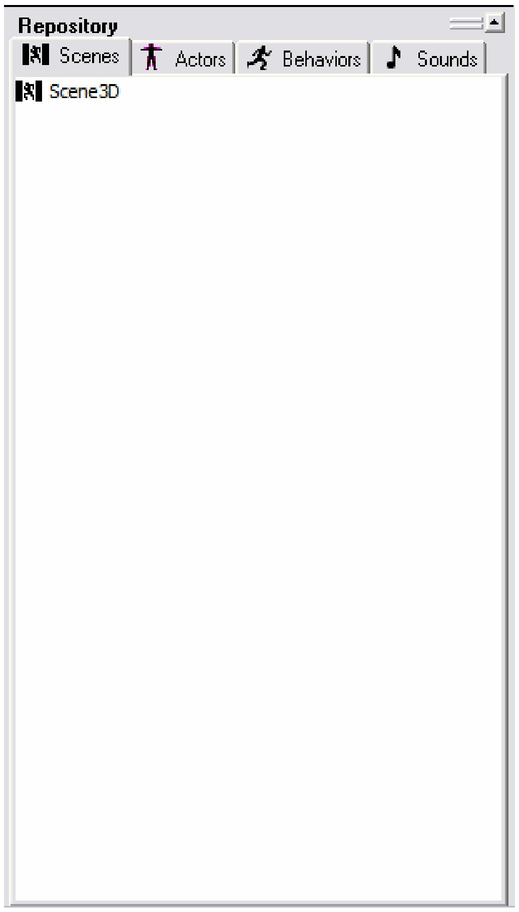
The Relational Event Dialog allows users to create events that monitor Attribute values. Attributes can be compared with relational operators like =, <, >, <=, >=, and !=. The Tree Controls present the user with a list or all properties and parameters in a production. To create a relational event test the user must specify a left hand term, a right hand term, and an operator. Optionally, the user may specify a constant for use as the right hand term.

Repository Window



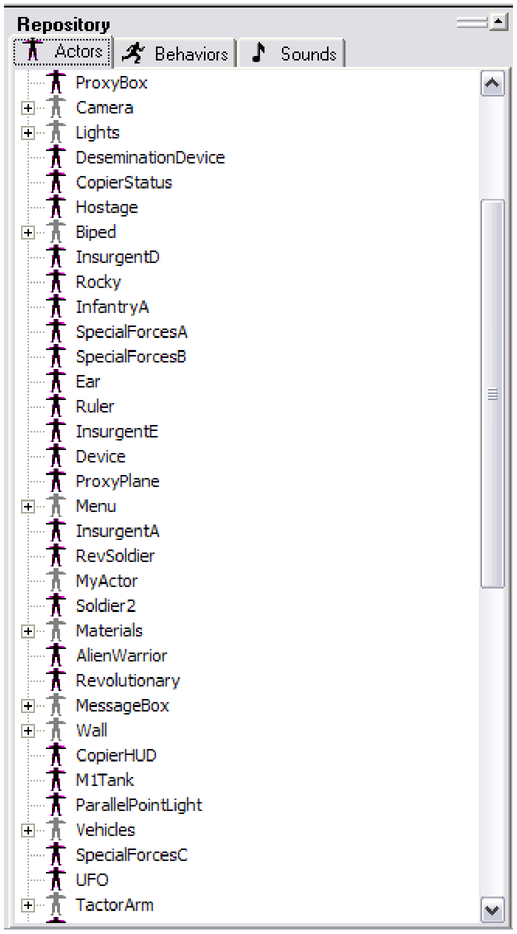
The Repository Window allows users to choose components for use in a Production. The Repository Window organizes components by tabs. Scenes, Actors, Behaviors, Events, and Sounds all have their own tabs. Each Component Tab has a slightly different way of organizing components. See the appropriate section for details on using these views. Components can be added to each of these tabs by right clicking and choosing the Add menu option. Components can be removed by right clicking on a component and choosing the Delete menu option. Adding and removing components in the Repository Window, also adds and removes the component from the local machine's component registry. In order to be used, a components must be registered with the component registry and be located in one the production's search paths. Path entries can be added from the Main Window's Production\Settings\ Search Paths menu. To use a component the user drags a component to the Production Window in the case of scenes, to the Scene Window for actors, the Behavior Window when it's a behavior or event, and to the Track Window in the case of sounds. The cursor will change from a stop sign to a go sign when the user is over a suitable drop target for the selected component. Releasing the mouse button will drop the component onto the view.

Scenes tab



The Scenes Tab displays the set of available scenes on the system. Scenes can be inserted into a production by dragging them to the Production Window and dropping them on the Production Item.

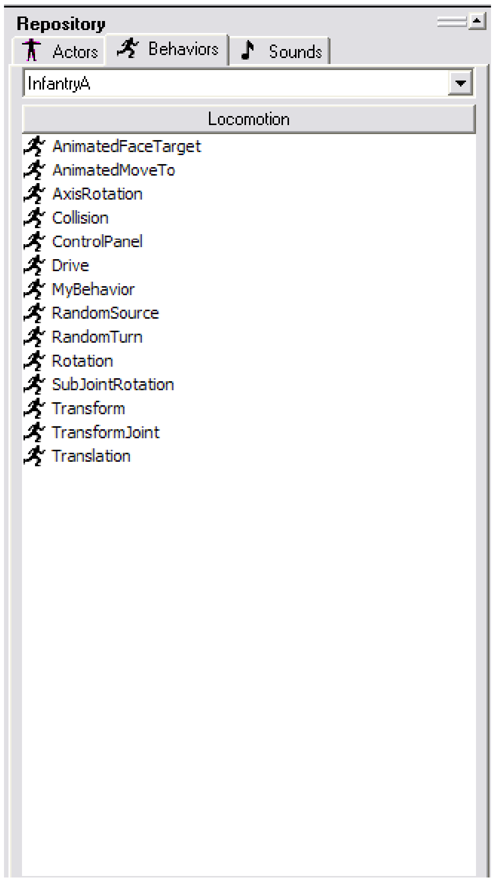
Actors Tab



ProxyBox
Camera
Lights
DeseminationDevice
CopierStatus
Hostage
Biped
InsurgentD
Rocky
InfantryA
SpecialForcesA
SpecialForcesB
Ear
Ruler
InsurgentE
Device
ProxyPlane
Menu
InsurgentA
RevSoldier
MyActor
Soldier2
Materials
AlienWarrior
Revolutionary
MessageBox
Wall
CopierHUD
M1Tank
ParallelPointLight
Vehicles
SpecialForcesC
UFO
TactorArm

The Actors Tab displays actors as a tree structure organized by ancestry. Actors can be inserted into a production by dragging them to the Production Window and dropping them on a Scene Item or by dropping them onto the Scene Window.

Behaviors Tab



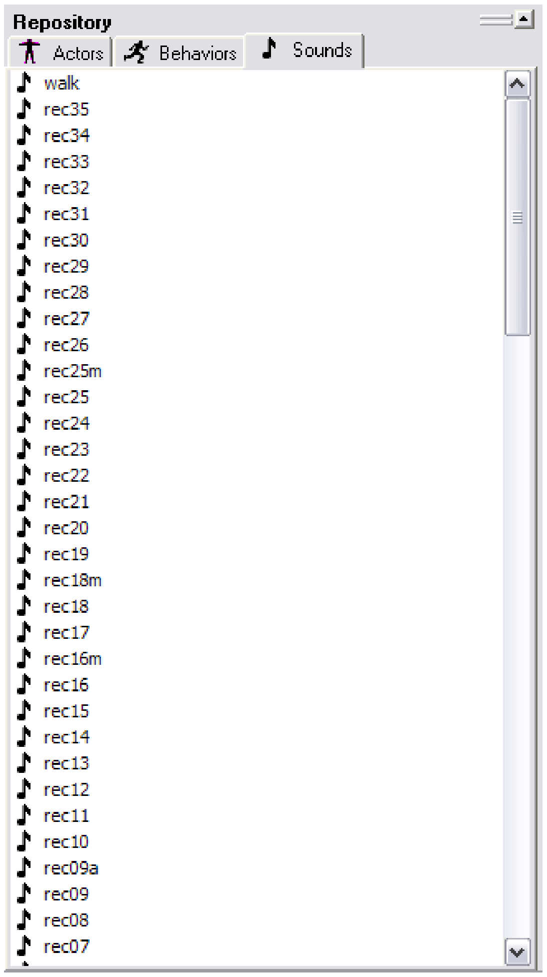
The Behaviors Tab displays behaviors as a list. The behavior list is filtered according to what actor's behavior graph is currently being edited in the Behavior Window. The Drop down list at the top of the tab allows the user to select any actor they wish as the filter. Additionally, behaviors are filtered by the Behavior Category button that allows the user to constrain the list to behaviors of a particular type. Behaviors can be inserted into a production by dragging them onto the Behavior Window and dropping them into the level of the hierarchy they are working on or they can drag them to the Production Window and drop them onto a behavior item.

Events Tab

Insert Picture Here

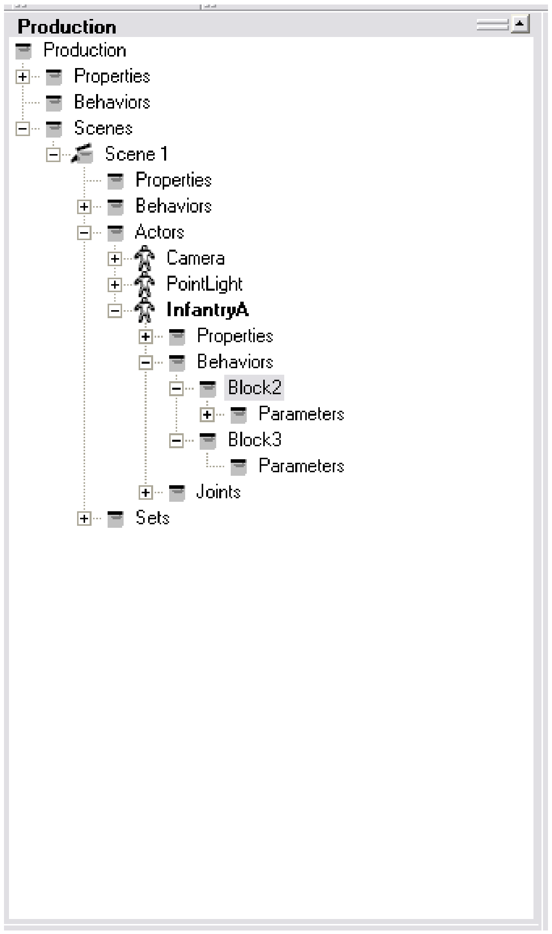
The Events tab displays events organized by type. Event can be dragged and dropped onto the start event or end event list of a behavior in the Behavior Window or onto the start and event items in the Production Window.

Sound Tab



The Sound tab displays all of the sounds registered in the system. Sounds can be dropped onto the Track Window's SoundFx track.

Production Window



The Production Window allows users to manage the contents of the current production as a hierarchical tree structure. All components that have been inserted into a production are visible from this window. Clicking on the "+" or "-"next to a tree item or double clicking the tree item expands the items beneath. Components dragged from the Repository Window are inserted into the production as sub-objects of the tree item they are dropped upon. Components can only be dropped on those items that are capable of containing them. The mouse cursor turns from a stop symbol to a go symbol over those items that are appropriate drop targets. Clicking on any component displays its attributes in the Attributes Window. Clicking on an actor has the additional effect of making it the currently selected actor in the Scene Window. Clicking on a behavior similarly makes it the currently selected behavior in the Behavior Window and navigates the view to display it. Each component has a unique set of options for its manipulation that can be accessed by right clicking on its respective tree item. Right clicking anything but the tree provides a set of menu options for adjusting the way in which the tree is displayed.

Production Tree

The Tree can be manipulated by right clicking on empty space in the Production Window.

<u>Menus</u>

Production Window Right Click Menu

The Table below describes the functionality of each menu item.

Flatten	Displays the behaviors of each entity as a flat alphabetical list as opposed to a hierarchical list.
Filter\All	A preconfigured filter that displays all possible elements according to the default setting.
Filter\Actors	A preconfigured filter that displays only actors.
Filter\Actors w/ Properties	A preconfigured filter that displays only actors and their properties.
Filter\Joints	A preconfigured filter that displays all actors and only their joints.
Filter\Joints w/ Properties	A preconfigured filter that displays all actors and their joints as well as the joint's properties.
Filter\Behaviors	A preconfigured filter that displays all actors and their behaviors.
Filter\Behaviors w/ Parameters	A preconfigured filter that displays all actors and their behaviors as well as the behavior's parameters.
Filter\Routes	A preconfigured filter that displays all actors and their behaviors as well as the behavior's routes.
Custom	Provides a list of elements in the production that can be filtered. Checked items will be displayed. The state of items on this list will change in response to the use of preconfigured filters.

Production Item Right Click Menu

A production can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Rename	Renames the production.
Add Property	Opens the Add Property Dialog.
Paste Scene	Places a copy of the scene clipboard contents within the current view.
Import Scene	Imports a scene from a file.
Add New Scene	Adds a new default scene to the production.

Scene Item Right Click Menu

A scene can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Cut	Removes the object from the view and places it in the clipboard.
Copy	Places a copy of the current selection in the clipboard.
Export	Exports the scene to a file.
Delete	Removes the current selection from the view.
Rename	Renames the scene.
Add Property	Opens the Add Property Dialog.
Paste Actor	Places a copy of the actor clipboard contents within the current view.
Import Actor	Import an actor from a file.
Set As Active Scene	Makes the currently selected scene active. The new scene's content will replace the previous scene's content in currently open windows.

Actor Item Right Click Menu

An actor can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Cut	Removes the object from the view and places it in the clipboard.
Copy	Places a copy of the current selection in the clipboard.
Export	Exports the actor to a file.
Delete	Removes the current selection from the view.
Rename	Renames the scene.
Add Property	Opens the Add Property Dialog.
Look Through	Applies to Cameras only. Binds this camera to the Scene Window view.

Property Item Right Click Menu

A property can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Change Type	Provides a list of available types. Type changes may result in a loss of data.
Delete	Removes the current selection from the view.

Behavior Item Right Click Menu

A behavior can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Add Parameter	Opens the Add Parameter Dialog.

Parameter Item Right Click Menu

A parameter can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Change Type	Provides a list of available types. Type changes may result in a loss of data.
Delete	Removes the current selection from the view.

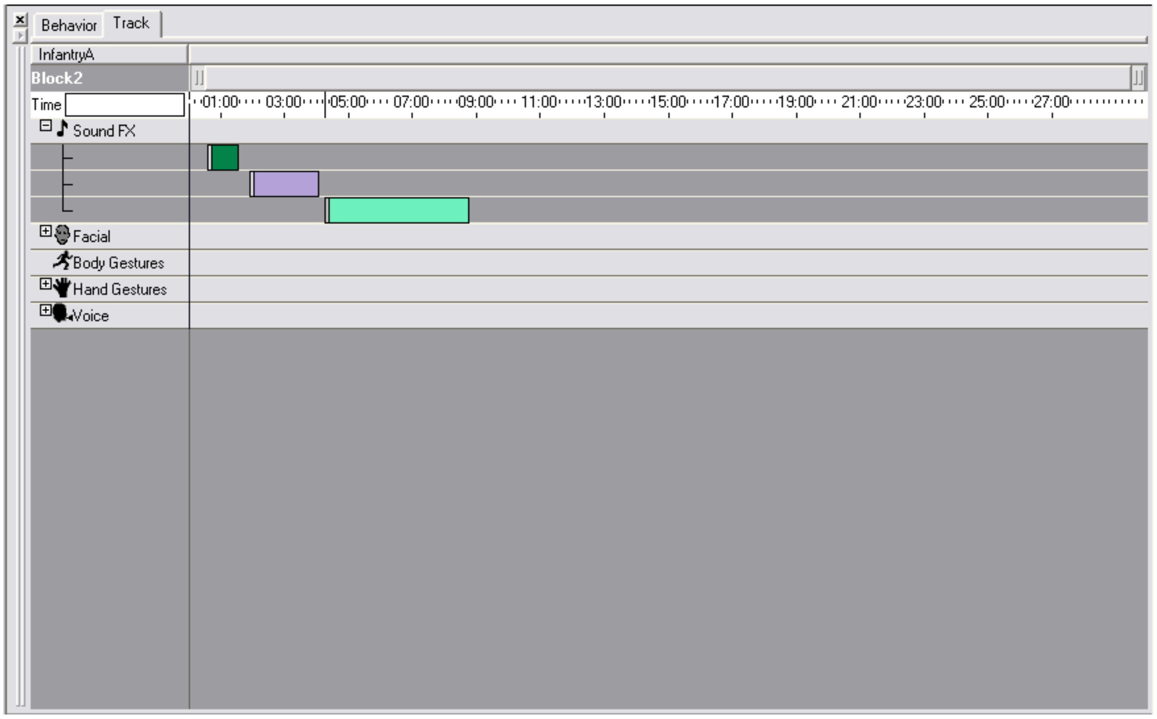
Route Item Right Click Menu

A route can be manipulated by right clicking on it in the Production Window.

The Table below describes the functionality of each menu item.

Delete	Removes the current selection from the view.

Track Window



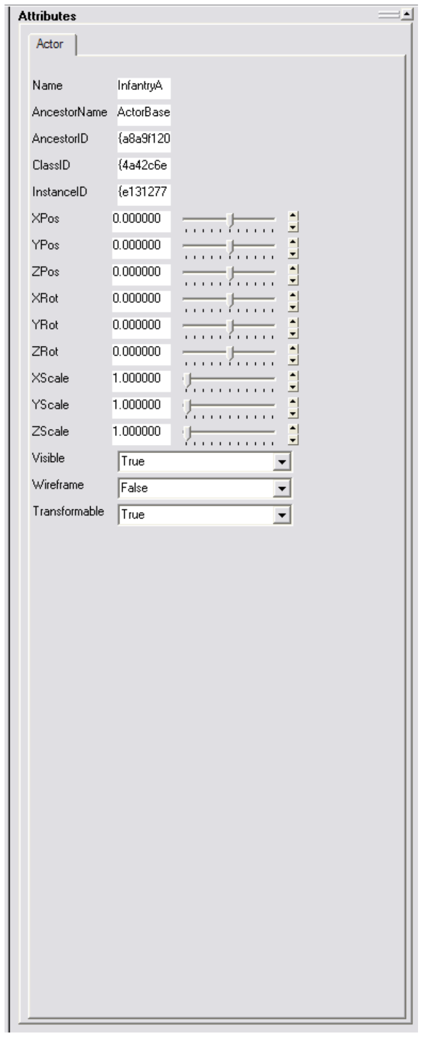
The Track Window lets the user layer and blend animation sequences and sound. Each behavior can have sounds and animations played when activated. When a behavior is selected in the Behavior Window the Track Window will display the set animations and sounds that are currently sequenced for that behavior. Animations and sounds are organized into Tracks; Tracks contain Clips which represent individual animation sequences. A Track can have multiple clips in sequence. Clips are independent, reusable sequences that are easy to merge or blend with other clips. Sound Clips can be dragged from the Repository Window and dropped on the SoundFx track. Animations can be dragged from the animation list of an Actor Item in the Production Window and dropped onto an animation track. Voice and waveforms can also be specified for sequencing the speech of an actor. Once a clip has been dropped onto a track, the clip can be manipulated by right clicking and changing its properties or by left clicking and dragging the clip around on the timeline. Clips can be lassoed and manipulated as a group as well. When a Behavior is active the Track Window will display the Play Line that shows what part of a sequence is playing at any particular moment in time. Additionally, the view can be scaled to better display lengthy sequences. Dragging the timelines scale manipulators can expand or contract the timeline. Dragging the scroll bar can advance the view back and forth in time.

Menus

<u>Track Right Click Menu</u>

Delete	Removes the current clip from the view.
Continue Playing After Behavior is Done	Allows a clip to continue playing to completion even after a behavior is done. The default behavior is to interrupt a clip when a behavior transitions to the done state.

Attributes Window



Name	InfantryA
AncestorName	ActorBase
AncestorID	{a8a9f120
ClassID	{4a42c6e
InstanceID	{e131277
XPos	0.000000
YPos	0.000000
ZPos	0.000000
XRot	0.000000
YRot	0.000000
ZRot	0.000000
XScale	1.000000
YScale	1.000000
ZScale	1.000000
Visible	True
Wireframe	False
Transformable	True

The Attributes Window displays the properties and parameters of an object in response to selection changes in other Windows. Attributes of an object are displayed as a list of name value pairs. Each attribute is presented with a distinct manipulator based on its type. The manipulator allows the user to quickly change the value of an attribute. The value of an attribute can also be changed by entering its value directly. Mousing over an attribute provides the user with a tool tip that presents additional information about an attribute such as its range, the displayed range, and its type. Routes can be created from properties to parameters by dragging a property from the Production Window and dropping it on a parameter. A type conversion warning will be issued if an automatic conversion cannot be applied to the route without data loss. This warning can be disabled from the right click menu for the view. Routes can be removed by right clicking on the attribute and choosing the Delete Route menu option.

Menus

Attributes Window Right Click Menu

Disable Route Warnings	Disables type conversion warnings when creating routes.

Attribute Right Click Menu

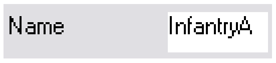
Locate	Scrolls the Production Window to display the property.
Delete Route (Only displayed when a parameter has a route bound to it)	Deletes the bounded route.

Route Drop Menu

By Value Once	Routes the value a single time when the behavior becomes active.
By Value Continuously	Routes the value every frame beginning when the behavior starts and ending when the behavior stops. Changes made to the value by the behavior are not reflected in the source property.
By Reference	Routes the value every frame beginning when the behavior starts and ending when the behavior stops. Changes made to the value by the behavior are reflected in the source property.
Use As Constant	Takes the current value of the property and uses it as the parameter value. No route is established.

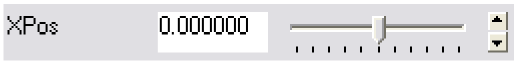
Manipulators

Text Manipulator



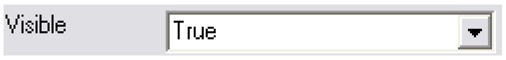
The text manipulator allows the direct entry of text. Text can be cut, copy and pasted via the right click menu.

Number Manipulator



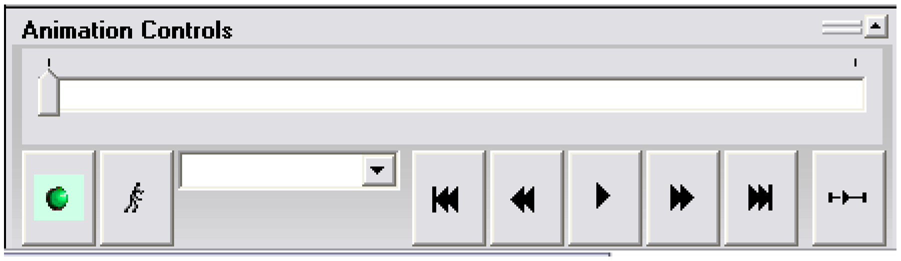
The number manipulator allows numbers (floats, ints, etc.) to be changed within a set range. For numbers with large ranges the up and down buttons to the right of the slider control will scale the displayed range for easier selection.

Enumeration Manipulator



The enumeration manipulator allows the user to choose from a finite set of elements presented as a combo box.

Animation Controls Window

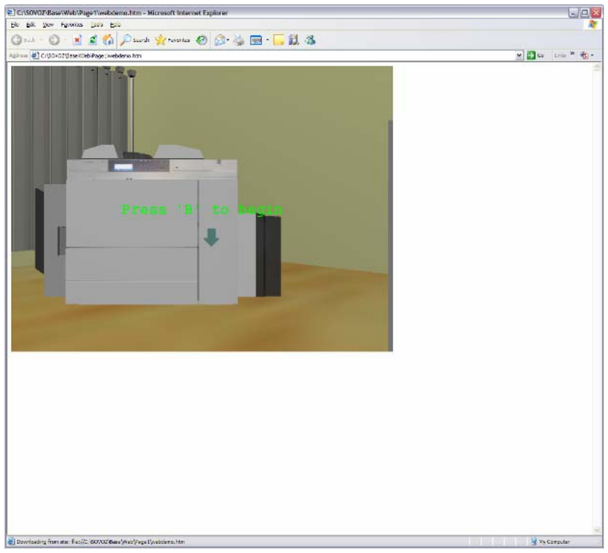


The Animation Controls Window allows behaviors to be stopped and started. The Play button when depressed will start the currently selected behavior in the Behavior Window. In the future, the other buttons that comprise the animation controls will be expanded to include more sophisticated control over the playback and recording of animation. Currently the additional buttons have no effect when pressed.

PROSCENA PLAYER

Once you have created a production you can deploy it via the Web or CD-ROM. Proscena provides two players that can be used to play back your production.

WEB PLAYER


C:\SOVOZ\Base\Web\Page1\webdemo.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Press 'B' to begin

The Web Player is an ActiveX Control that can be embedded in a Web Page to execute productions. It can be found in the <InstallLocation>\Bin directory and is named SzWebPlayer.dll. The Web Player is registered on the machine by default at installation time. The <InstallLocation>\SDK\WebDemo directory contains an HTML file called webdemo.htm that demonstrates its use. Note: webdemo.htm expects a production called index.szp to be present in the directory from which it is launched.

Sample

The Web Player provides facilities for downloading a production and its associated components to the local machine and executing them. The following code provides a minimal web page that can be used for loading the control.

```
<html>
<object id="EngineLoaderControl" classid="CLSID:C43024CA-BDF9-4ab3-91E1-16E63926A11B" CODEBASE="engine.cab">
<PARAM NAME="szArchiveFileName" VALUE="index.szp">
<PARAM NAME="szDataPath" value="c:\progra~1\sovoz\proscena\bin">
<PARAM NAME="Left" value="125">
<PARAM NAME="Top" value="10">
<PARAM NAME="Bottom" value="1025">
<PARAM NAME="Right" value="1575">
<PARAM NAME="szSourceLocation" value="http:\\www.sovoz.com\test\engine">
</object>

<script language="JScript">
function window.onunload() { EngineLoaderControl.Kill(); }
</script>
</html>
```

The following table describes the usage of the parameters used in the <OBJECT> tag.

SzArchiveFileName	Path of the production.
szDataPath	The default path to be provided to control. This is overridden by any path settings provided in the production.
Left	The left extent of the window in screen coordinates.
Top	The top extent of the window in screen coordinates.
Bottom	The bottom extent of the window in screen coordinates.
Right	The right extent of the window in screen coordinates.
szSourceLocation	The remote location of the engine and any components needs to load. The production. (Not needed for local use)

DESKTOP PLAYER

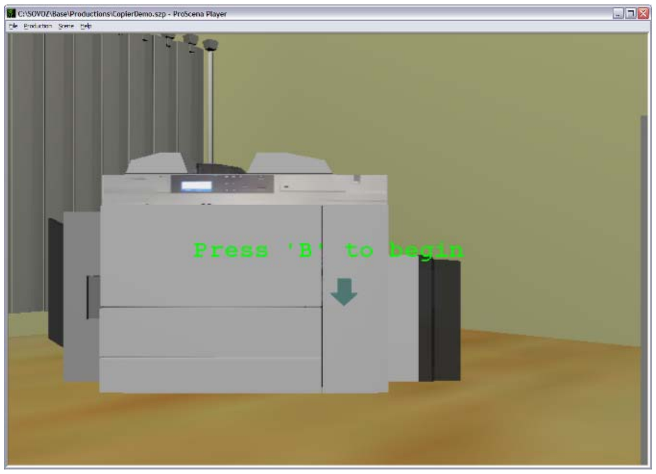

C:\SOVOZ\Base\Productions\CopierDemo.szp - ProScena Player
File Production Scene Help
Press 'B' to begin

Figure xx. ProScena Desktop Player

The Desktop Player is a standalone executable that can be used to execute productions. It can be found in the <InstallLocation>\Bin directory and is named SzDesktopPlayer.exe

The Main Window provides the user with a menu with options for opening and configuring a production.

Menus

File Menu

The File Menu has commands for opening and saving a production to the file system. The Settings\Input Devices menu allows the user to modify the device mappings on his local computer.

Open	Opens an existing Production.
Save	Saves the current Production in a *.szp* file.
Save As	Saves the current Production with a new name.
Settings\Input Devices	Opens the Input Devices Dialog. Allows users to define custom action maps for the production.
Exit	Closes the current Production and exits the application.

Production Menu

The Production Menu contains commands for managing the current production. The Settings\Search Paths menu option launches the Search Paths dialog that allows the user to change the paths used by the system to locate components for the current production. Any path changes made will be saved in to the production and used on subsequent loads.

Settings\Search Paths	Opens the Search Paths Dialog.

Scene Menu

The Scene Menu has options for configuring the currently active scene. The Renderer menu option contains submenus for changing the shading, anti-aliasing, and full screen options of the Renderer.

Renderer\Set Shading Mode	Sets the scenes shading as either flat or smooth and either point, wireframe, or solid.
Renderer\Set Antialiasing Mode	Sets the scenes antialiasing from None up to 16x.
Renderer\Set Full Screen Mode	Displays the scene in one of the full screen modes available on the system.

Help Menu

The Help Menu provides entry to the help system, information about Sovoz, and version information about the authoring tool. Detailed context sensitive help is available anywhere in the player by pressing F1.

Contents	Opens the help system to the Table of Contents page.
Index	Opens the help system to the Index page.
Search	Opens the help system to the Search page.
Sovoz on the Web	Opens a web browser to the Sovoz corporate web page.
About Proscena	Opens the About dialog.

APPENDIX B – SCRIPTING REFERENCE

Entities, Behaviors and Events can be written in either C++ or Python.

SCRIPTABLE OBJECTS

Objects are manipulated by setting their properties and calling their methods. Scripts can be added to a production via the authoring tool. See the Script Event and Script Behavior discussions in the authoring tool section for more details. The following properties and methods are scriptable.

Behavior

Behavior Properties

Note: when accessing these properties use the mBehavior variable.

owner	Returns the entity that owns this behavior
Container	Returns the behavior's parent.
MetaBehavior	Returns the meta-behavior of this behavior.
IsActive()	A boolean indicating whether the behavior is active.
IsIdle()	A boolean indicating whether the behavior is idle.
IsDone()	A boolean indicating whether the behavior is done.
IsPendingDone()	A boolean indicating whether the behavior is pending done.
Name	The name of this behavior
<Sub-behavior name>	The sub-behavior with that name.
<Event name>	The event with that name.
<Parameter name>	The parameter with that name.

Behavior Methods

Init()	A script executed only once, when the behavior's starting event first evaluates to true, that initializes internal behavior variables, sensors, focus, etc. and possibly sets external variable and event values.
Go()	A script executed once per active cycle every time a behavior's starting event evaluate to true.
Sense()	The sense phase is where the behavior retrieves the state of the world. It is critically important that behavior writers be in the habit of sampling production state only in sense so that entities stay in sync.
Control()	The control phase is where the behavior computes what actions it will take based upon sensor measurements and other information obtained during the sense phase. The control script is where the details of these actions are specified. These actions can be scripted using various drivers: motion data, analytic curves (ramps, splines, etc.), other behaviors and custom program logic.
Act()	The act phase is where the behaviors changes should be applied to the entity and the entity's state is updated in the production. If the behavior itself has dynamics, appropriate code must be added to the act script so that these dynamics can be evaluated.
Terminating()	A script executed only once, when the behavior ending event is triggered, that initializes internal behavior variables, sensors, focus, etc. and possibly sets external variable and event values.

Entity

Entity Properties

<Joint name>	The joint object with that name.
<Property name>	The property object with that name.

Entity Methods

Note: The following methods are for future use. Only Behaviors and Events can have their methods scripted in Python. See Appendix C for details about scripting Actors.

Init()	A script executed only once, when the actor is first loaded, can be used to initialize properties, etc.
Sense()	- The sense phase is where the entity samples the state of the world. It is critically important that entity writers be in the habit of sampling production state only in sense so that entities stay in sync.
Control()	The control phase is where the entity computes what actions it will take based upon sensor measurements and other information obtained during the sense phase.
Act()	The act phase is where the entity should save its state to its properties. If the behavior itself has dynamics, appropriate code must be added to the act script so that these dynamics can be evaluated.

Joint

Joint Properties

abs	The absolute position of the joint.
rel	The relative position of the joint.
isRoot	A boolean indicating whether the joint is the root of the hierarchy.
<Joint name>	The sub-joint with that name.

Event

Event Methods

Note: to access the owning behavior use the mBehavior variable.

Test()	A Script executed when an event is tested. This script can perform any number of calculations but must return True or False.

APPENDIX C – .ACT AND .BOD REFERENCE

Creating an Actor does not necessarily have to involve writing C++ or Python code. Two small description languages have been devised to allow people to add actors to the system without writing code. To add ANY actor to the system two files are required an .Act and .Bod file. The .Act file provides metadata about the actor and its ancestry as well as a pointer to the .Bod file that it uses to describe the joints of the actor. The .Bod file contains a specification of the actor's joint hierarchy. The <InstallLocation>\Actors directory contains numerous examples of how to construct these files.

.ACT FILE

.Act files can be created in a simple text editor. The following is an example of a Chevrolet Corvette actor.

```
ancestor   Actor
ancestor   Vehicle
ancestor   ChevroletCorvette
setname    ChevroletCorvette
readfile   ChevroletCorvette.bod
ancestorUid {35779DD5-8181-4129-888A-89D7AE6A6E79}
uid        {52A06037-FA06-47ab-A655-4103254F3D1E}
```

This file uses name-value pair syntax. Each line starts with a name and is followed by a space and a value. The following table describes the names and expected values for this file.

Name Value Pairs

ancestor	The ancestor name-value pair specifies the name of the component that this actor inherits from. The entire ancestry must be specified in order to properly register an actor in the component repository. An abstract component type must be defined in the .act file and that ancestor component must be registered. Typically, the base abstract component type Actor is all that is required. This file happens to reference a derived actor as well.
setname	The setname name-value pair specifies the name of this component.
readfile	The readfile name-value pair references the .bod file used for this actor.
ancestorUid	The ancestorUid name-value pair specifies the direct ancestor's component unique id (Uid). Unlike the ancestor name-value pair only the most direct ancestor need be specified. This information should come with the component documentation or can be

	found in the source files.
uid	The uid specifies the unique id of this component. Use the <InstallLocation>\SDK\3p\GuidGen\Guidgen.exe tool to generate the Uid.

.BOD FILE

.Bod files can also be created in a simple text editor. The following is an example of the body for the Chevrolet Corvette actor.

```
body
{

 joint   root
  {
          graphic -1
  }

newjoint ChevroletCorvetteBody {
          parent root
          graphic ChevroletCorvetteBody
          preset { pos 0.0, -20.0, 0.0 }
          preset { angd -90 0 0 } }
 }

newjoint LeftBlinker {
           parent ChevroletCorvetteBody
           graphic ChevroletCorvetteLeftBlinker

}

newjoint RightBlinker {
          parent ChevroletCorvetteBody
          graphic ChevroletCorvetteRightBlinker
 }

setcontact root

}
```

This file uses a slightly more complicated syntax. The file is divided into scopes that in turn define name-value pairs. It declares four joints the root, ChevroletCorvetteBody, LeftBlinker, and RightBlinker. The ChevroletCorvetteBody is parented to the root and the Left and Right Blinkers are parented to the ChevroletCorvetteBody . The graphic name-value pair tells the system which graphic file to load to represent the joint. The following table describes the scopes and name-value pairs that are expected.

Scopes

body	The body scope defines the boundaries of a joint hierarchy definition. There will be only one scope per .bod file and that scope should include all other scopes within it. This is a top level scope and occurs only once in a file.
joint	The joint scope is a special scope and occurs only once at the beginning of the body scope. It is intended to identify the root of the joint hierarchy. The value of the joint scope should always be root. This scope resides inside body scope.
newjoint <jointname>	The newjoint scope defines a new joint of name <jointname>. This scope resides inside body scope.
newmanagedjoint <jointname>	The newmanagedjoint scope defines a new joint of name <jointname>. This joint is distinct and is only used for joint's that reference an actormanager. This scope resides in the body scope.
preset	Specifies a preset transformation for a joint. This scope resides inside a joint scope.

Name-Value Pairs

parent	The parent name-value pair identifies the joint's parent and should reference a previously defined joint scope. This name-value pair resides inside a joint, newjoint, or newmanagedjoint scope.
graphic	The graphic name-value pair identifies the .nif file that should be loaded to represent that joint in the scene. Note: Do not append the .nif extension when referencing the file. Specifying a -1 means no graphic should be loaded. This name-value pair resides inside a joint, newjoint, or newmanagedjoint scope.
angd	A XYZ tuple specifying rotation in degrees. This name-value pair is found inside a preset scope.
pos	A XYZ tuple specifying position in world units. This name-value pair is found inside a preset scope.
setcontact	The setcontact name-value pair defines the contact point for the actor and is only used for actors that employ IK. All other actors should set this value to refer to the root scope. This name-value pair resides inside the body scope.

APPENDIX D – COMPONENT REFERENCE

The following reference describes the properties and parameters of the components included in these libraries. The source code for the components can be found in the <InstallLocation>\SDK directory. Note: C++ components can be registered in the component repository by either using the SzRegister.exe utility located in the <InstallLocation>\Bin directory (use the -? option for instructions.) or by using the Repository Windows Right Click "Add" menu option.

COMMON COMPONENTS

The Common Components that follow are realized in the CommonComponents.dll file.

Menu

The menu actor supports common functionality of a typical user menu. The menu is made up of menu items, which is a set of three groups of text and images. Menu items can have nested menu items themselves. A formatted string (see below for format) is used to represent the set of menu items, and the menu can be created from this string by entering it in the MenuString property. Most properties on the Menu Actor affect formatting, modification, and visibility of all menu items in the menu. To affect only one menu item, use the set of 'Action' properties.

Menu Format

MENUITEM = { "name", "left_image_filename", "left_selected_image_filename", left_image_U0, left_image_V0, left_image_U1, left_image_V1, "left_text", "middle_image_filename", "middle_selected_image_filename", middle_ image_U0, middle_ image_V0, middle_ image_U1, middle_ image_V1, "middle_text", "right_image_filename", "right_selected_image_filename", right_ image_U0, right_ image_V0, right_ image_U1, right_ image_V1, "right_text", visible, enabled, { comma seperated list of MENUITEM } }

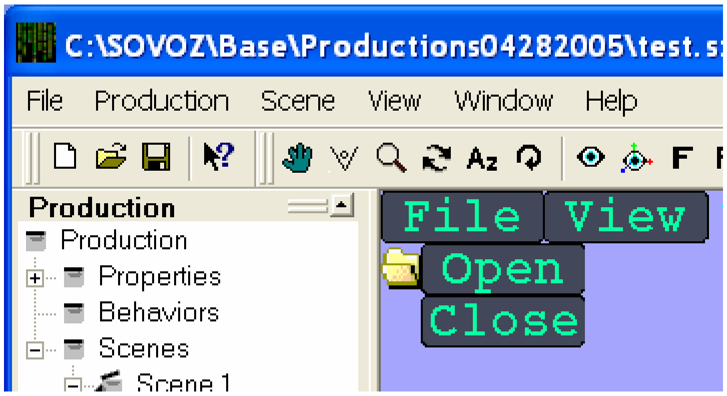
MENU = { comma seperated list of MENUITEM }

The U0,V0,U1,V1 portions of the menu item string represent the lower left and upper right texture coordinates of the menu item respectively. The 'image_filename' in the menu item string is the main image that the menu item displays. The 'selected_image_filename' in the string is the image displayed when that menu item is selected with the mouse by the user. All filenames must be in the production's path. The 'visible' part of the string can have a value of 1 (visible in the menu) or 0 (not visible). The 'enabled' part of the string can have a value of 1 (allowed to be selected) or 0 (not selectable).

Menu Example

Here is an example of a menu string and the menu it produces:

```
{{"File","","",,,,,"","MiddleMenuItem.tga","",0.0,0.0,0.543,1.0,"File","","",,,,,"",1,1,{{"Open
","FileOpen.bmp","",,,,,"","MiddleMenuItem.tga","",0.0,0.0,0.543,1.0,"Open","","",,,,,"",1,1,
{}},{"Close","","",,,,,"","MiddleMenuItem.tga","",0.0,0.0,0.543,1.0,"Close","","",,,,,"",1,1,{}
}}},{"View","","",,,,,"","MiddleMenuItem.tga","",0.0,0.0,0.543,1.0,"View","","",,,,,"",1,1,{}
}}
```



The table below describes the functionality of each parameter.

<Parameter>	<Description>
Enabled	True to interact with the menu. False to disable any menu interactions.
Units	Normalized – Properties below will be interpreted in normalized screen amounts. Values range from 0 (zero percent of the screen) to 1 (100 percent of the screen). Pixels – Properties below will be interpreted in pixel amounts.
SizeMode	Fixed – Use the specified widths and heights for the size of all menu items. Auto – Uses the specified widths and heights as a minimum for the size of all menu items, but expands the size if the images or text need more room.

ItemLeftWidth	The width of the left part of a menu item as interpreted by the SizeMode and Units properties.
ItemMiddleWidth	The width of the middle part of a menu item as interpreted by the SizeMode and Units properties.
ItemRightWidth	The width of the right part of a menu item as interpreted by the SizeMode and Units properties.
ItemHeight	The height of a menu item as interpreted by the SizeMode and Units properties.
TopLevelAlign	Top – The top level menu is positioned from left to right at the top of the screen and the sub menus are positioned underneath. Left – The top level menu is positioned from top to bottom at the left of the screen and the sub menus are positioned to the right. Right – The top level menu is positioned from top to bottom at the left of the screen and the sub menus are positioned to the left. Bottom – The top level menu is positioned from left to right at the bottom of the screen and the sub menus are positioned on top.
TopLevelHorizOffset	The horizontal offset from the left of the screen where the menu will be displayed, as interpreted by the Units property.
TopLevelVertOffset	The vertical offset from the top of the screen where the menu will be displayed, as interpreted by the Units property.
TopLevelLastSelect	Specifies whether the last selected top level menu item should remain selected until another selection occurs.
SubMenuPopupMode	Auto – When a menu item is selected in a submenu, that menu item's submenu (if any) will automatically position itself. Replace – When a menu item is selected in a submenu, that menu item's submenu (if any) will replace the current visible submenu.
SubMenuActivationMode	Specifies how non top level submenus should be displayed. They can be displayed when the mouse click's on an item or when the mouse is over an item.

MouseOverTimeDelay	If the SubMenuActivationMode is set to MouseOver, this is the amount of time the mouse must be over a non top level menu item before it's submenu is displayed.
Font	The font to use for all text in the menu.
TextHorizMargin	The horizontal offset from the left of the left, middle, and right parts of the menu item where the text will be displayed, as interpreted by the Units property.
TextVertMargin	The horizontal offset from the left of the left, middle, and right parts of the menu item where the text will be displayed, as interpreted by the Units property.
BordersEnabled	Specifies if borders should be displayed.
TopLevelBordersEnabled	Specifies if the top level borders should be displayed.
BackPanelEnabled	Specifies if the background is visible for all menu items.
TopBorderImage	The filename of the image file to use for the top border image. This will be used as the top border for the top level menu and all sub menus.
TopBorderHeight	The height of the top border as interpreted by the Units and SizeMode properties.
TopBorderU0	The U component of the lower left texture coordinate to apply to the top border image.
TopBorderV0	The V component of the lower left texture coordinate to apply to the top border image.
TopBorderU1	The U component of the upper right texture coordinate to apply to the top border image.
TopBorderV1	The V component of the upper right texture coordinate to apply to the top border image.
BottomBorderImage	The filename of the image file to use for the bottom border image. This will be used as the bottom border for the top level menu and all sub menus.
BottomBorderHeight	The height of the bottom border as interpreted by the Units and SizeMode properties.

BottomBorderU0	The U component of the lower left texture coordinate to apply to the bottom border image.
BottomBorderV0	The V component of the lower left texture coordinate to apply to the bottom border image.
BottomBorderU1	The U component of the upper right texture coordinate to apply to the bottom border image.
BottomBorderV1	The V component of the upper right texture coordinate to apply to the bottom border image.
LeftBorderImage	The filename of the image file to use for the left border image. This will be used as the left border for the top level menu and all sub menus.
LeftBorderWidth	The width of the left border as interpreted by the Units and SizeMode properties.
LeftBorderU0	The U component of the lower left texture coordinate to apply to the left border image.
LeftBorderV0	The V component of the lower left texture coordinate to apply to the left border image.
LeftBorderU1	The U component of the upper right texture coordinate to apply to the left border image.
LeftBorderV1	The V component of the upper right texture coordinate to apply to the left border image.
RightBorderImage	The filename of the image file to use for the right border image. This will be used as the right border for the top level menu and all sub menus.
RightBorderWidth	The width of the right border as interpreted by the Units and SizeMode properties.
RightBorderU0	The U component of the lower left texture coordinate to apply to the right border image.
RightBorderV0	The V component of the lower left texture coordinate to apply to the right border image.
RightBorderU1	The U component of the upper right texture coordinate to apply to the right border image.

RightBorderV1	The V component of the upper right texture coordinate to apply to the right border image.
SubMenuArrow	The filename for an arrow image to be used when a menu item contains a submenu.
SubMenuArrowSelected	The filename for an arrow image to be used when a menu item contains a submenu and when that menu item is currently selected.
ItemBackgroundR	The red component of the background color to apply to all menu items. (values range from 0 to 1)
ItemBackgroundG	The green component of the background color to apply to all menu items. (values range from 0 to 1)
ItemBackgroundB	The blue component of the background color to apply to all menu items. (values range from 0 to 1)
TextColorR	The red component of the text color to apply to all menu items. (values range from 0 to 1)
TextColorG	The green component of the text color to apply to all menu items. (values range from 0 to 1)
TextColorB	The blue component of the text color to apply to all menu items. (values range from 0 to 1)
Action	Use the action property in conjunction with the properties below by choosing the action first, then setting the appropriate properties. Destroy – This deletes a menu item from the menu. Enable – This makes the menu item selectable. Disable – This makes the menu item unselectable. Hide – This causes the menu item to not be displayed. Show – This causes the menu item to be displayed. Select – This selects the menu item as if the user selected it. Deselect – This deselects the menu item as if the user deselected it. To perform any of the above actions, first choose the action, then fill in the ActionMenuItem property with the menu item to perform the action on. SetLeftText – This sets the text of the left part of a menu item.

	SetMiddleText – This sets the text of the middle part of a menu item. SetRightText – This sets the text of the right part of a menu item. To perform any of the above text setting actions, first choose the action and fill in the ActionValue with the text to use, then fill in the ActionMenuItem property with the menu item to perform the action on.
ActionMenuItem	The action menu item specifies which menu item to apply an action to. The menu item is specified by name with a dotted string notation also containing the names of all parents. (For example, to specify a submenu item, a name such as TopLevelItem1_Name.ChildItem3_Name… might be used.) The action specified by the Action property is performed after this parameter is filled in.
ActionValue	The action value specifies the value of the text to set when performing one of the SetXText actions. This should be filled in before the ActionMenuItem is filled in.
CreateMenuItem	A string representation of the menu item to be created. This should be filled in after filling in the MenuItemRef and after choosing the InsertAs properties.
MenuItemRef	The name of the menu item which serves as a place holder for where to insert a new CreatedMenuItem.
InsertAs	FirstChild Of MenuItemRef – Specifies that the menu item specified by CreateMenuItem should be inserted as the first child of the menu item specified by MenuItemRef. LastChild Of MenuItemRef – Specifies that the menu item specified by CreateMenuItem should be inserted as the last child of the menu item specified by MenuItemRef. Before MenuItemRef – Specifies that the menu item specified by CreateMenuItem should be inserted before the menu item specified by MenuItemRef. After MenuItemRef – Specifies that the menu item specified by CreateMenuItem should be inserted after the menu item specified by MenuItemRef.

MenuString	The string representation of the menu (see format above). The menu will reflect any changes made to this string. Any actions that modify the menu will also change the string.
FullSelectedItem	When the user selects a menu item from the menu, the full name in dotted string notation will be output to this property. It will be replaced when another selection is made.
SelectedItem	When the user selects a menu item from the menu, the name of the selected item (as specified in the menu string) will be output to this property. It will be replaced when another selection is made.

MessageBox

The message box actor is used to display a message to the user. It can contain a background image, text, borders, and a button. When the user clicks the button, the MessageBox will no longer be visible.

The table below describes the functionality of each parameter.

<Parameter>	**<Description>**
Units	Normalized – Properties below will be interpreted in normalized screen amounts. Values range from 0 (zero percent of the screen) to 1 (100 percent of the screen). Pixels – Properties below will be interpreted in pixel amounts.
HorizOffset	The amount that the center of the message box will be offset horizontally, as interpreted by the Units property. With a default of zero the message box appears in the middle of the screen.
VertOffset	The amount that the center of the message box will be offset horizontally, as interpreted by the Units property. With a default of zero the message box appears in the middle of the screen.
SizeMode	Fixed – Uses the specified widths and heights for the size of the message box. Auto – Uses the specified widths and heights as a minimum for the size of the message box, but expands the size if the image or text needs more room.
BackgroundTexture	The filename of the image to use for the background of this messagebox. This file must be in the path.
BackgroundWidth	The width of the message box as interpreted by the Units and SizeMode properties.
BackgroundHeight	The height of the message box as interpreted by the Units and SizeMode properties.
BackgroundR	The red component of the message box's background color. (values range from 0 to 1)

BackgroundG	The green component of the message box’s background color. (values range from 0 to 1)
BackgroundB	The blue component of the message box’s background color. (values range from 0 to 1)
BackgroundU0	The U component of the lower left texture coordinate to apply to the background image.
BackgroundV0	The V component of the lower left texture coordinate to apply to the background image.
BackgroundU1	The U component of the upper right texture coordinate to apply to the background image.
BackgroundV1	The V component of the upper right texture coordinate to apply to the background image.
Font	The font to use for the text in this message box.
Text	The text to display in the message box.
TextR	The red component of the message box’s text color. (values range from 0 to 1)
TextG	The green component of the message box’s text color. (values range from 0 to 1)
TextB	The blue component of the message box’s text color. (values range from 0 to 1)
MaxTextWidth	The amount of space (as interpreted by the Units property) that the text has to display before wrapping to the next line.
TextHorizMargin	The amount of space (as interpreted by the Units property) to the left and right of the text area.
TextVertMargin	The amount of space (as interpreted by the Units property) on top and bottom of the text area.
TopBorderImage	The filename of the image file to use for the top border image.
TopBorderHeight	The height of the top border as interpreted by the Units and SizeMode properties.
TopBorderU0	The U component of the lower left texture coordinate

	to apply to the top border image.
TopBorderV0	The V component of the lower left texture coordinate to apply to the top border image.
TopBorderU1	The U component of the upper right texture coordinate to apply to the top border image.
TopBorderV1	The V component of the upper right texture coordinate to apply to the top border image.
BottomBorderImage	The filename of the image file to use for the bottom border image.
BottomBorderHeight	The height of the bottom border as interpreted by the Units and SizeMode properties.
BottomBorderU0	The U component of the lower left texture coordinate to apply to the bottom border image.
BottomBorderV0	The V component of the lower left texture coordinate to apply to the bottom border image.
BottomBorderU1	The U component of the upper right texture coordinate to apply to the bottom border image.
BottomBorderV1	The V component of the upper right texture coordinate to apply to the bottom border image.
LeftBorderImage	The filename of the image file to use for the left border image.
LeftBorderWidth	The width of the left border as interpreted by the Units and SizeMode properties.
LeftBorderU0	The U component of the lower left texture coordinate to apply to the left border image.
LeftBorderV0	The V component of the lower left texture coordinate to apply to the left border image.
LeftBorderU1	The U component of the upper right texture coordinate to apply to the left border image.
LeftBorderV1	The V component of the upper right texture coordinate to apply to the left border image.
RightBorderImage	The filename of the image file to use for the right

	border image.
RightBorderWidth	The width of the right border as interpreted by the Units and SizeMode properties.
RightBorderU0	The U component of the lower left texture coordinate to apply to the right border image.
RightBorderV0	The V component of the lower left texture coordinate to apply to the right border image.
RightBorderU1	The U component of the upper right texture coordinate to apply to the right border image.
RightBorderV1	The V component of the upper right texture coordinate to apply to the right border image.
UseButton	Specifies if a button should be displayed or not.
ButtonR	The red component of the message box's button color. (values range from 0 to 1)
ButtonG	The green component of the message box's button color. (values range from 0 to 1)
ButtonB	The blue component of the message box's button color. (values range from 0 to 1)
ButtonText	The text to display in the button.
ButtonTexture	The filename of the image to use for the button.
ButtonSelectedTexture	The filename of the image to use for the button when it is selected. (Not Supported)
ButtonWidth	The width of the button as interpreted by the Units property.
ButtonHeight	The height of the button as interpreted by the Units property.
ButtonU0	The U component of the lower left texture coordinate to apply to the button image.
ButtonV0	The V component of the lower left texture coordinate to apply to the button image.
ButtonU1	The U component of the upper right texture coordinate

	to apply to the button image.
ButtonV1	The V component of the upper right texture coordinate to apply to the button image.

ActorLifecycle

Inserts or Removes and actor from the specified Scene.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
Mode	Specifies whether to add or remove the described Actor.
Scene	The name of the Scene where the Actor should be located.
Actor Class	The class name of the registered Actor to add.
Actor Name	The name the Actor will have when added, or the name of the Actor to remove.

ActorMonitor

Outputs world space positional and rotational information for an actor.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
X Pos	The x position of the actor.
Y Pos	The y position of the actor.
Z Pos	The z position of the actor.
X Rot	The x rotation (in radians) of the actor.
Y Rot	The y rotation (in radians) of the actor.
Z Rot	The z rotation (in radians) of the actor.
Direction X	The x direction of the actor.
Direction Y	The y direction of the actor.
Direction Z	The z direction of the actor.
Run	Once – When the behavior runs, it will output the above information and then stop. Continuously – The behavior will run and output the above information continuously.

AnimatedFaceTarget

Causes an actor to face a target while animating an actor with an animation manager.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`manager`	The animation manager to use.
`CurrentAnim`	The animation sequence that the actor is currently playing.
`frequency`	The speed of the animation sequence.
`phase`	The phase of the animation sequence.
`Accumulate`	True – The actor will accumulate any movement that occurs during an animation sequence. False – The actor will start from the animation sequence's original position every time the animation is run.
`Backwards`	True if the animation sequence should be played backwards.
`EndMode`	Stop Anim – The animation sequence will stop when the behavior stops. Continue – The animation sequence will continue even after the behavior stops.
`Looping`	On if the animation sequence should repeat itself when finished.
`Layering`	On if this animation sequence should be blended on top of another animation sequence. There should be at least one animation manager running with layering off. Any additional animation managers may have layering on.
LeftAnim	The animation to play while rotating left towards the target.
RightAnim	The animation to play while rotating right towards the target.

IdleAnim	The animation to play after facing the target.
xTarget	The x position of the target.
yTarget	The y position of the target.
zTarget	The z position of the target.
yaTolerance	The y rotational threshold that an actor should be within before it is considered to be facing the target.
turnSpeed	The speed at which the actor should turn while facing the target.
moveMode	FaceTarget – The behavior will run continuously causing the actor to face the target even after it moves. FaceTargetAndStop – The behavior will stop after it successfully faces the target.

AnimatedMoveTo

Causes an actor to move towards a target while animating an actor with an animation manager.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`manager`	The animation manager to use.
`CurrentAnim`	The animation sequence that the actor is currently playing.
`frequency`	The speed of the animation sequence.
`phase`	The phase of the animation sequence.
`Accumulate`	True – The actor will accumulate any movement that occurs during an animation sequence. False – The actor will start from the animation sequence's original position every time the animation is run.
`Backwards`	True if the animation sequence should be played backwards.
`EndMode`	Stop Anim – The animation sequence will stop when the behavior stops. Continue – The animation sequence will continue even after the behavior stops.
`Looping`	On if the animation sequence should repeat itself when finished.
`Layering`	On if this animation sequence should be blended on top of another animation sequence. There should be at least one base animation running with layering off. Any additional animation managers may have layering on.
FowardAnim	The animation to play while moving forward towards the target.
IdleAnim	The animation to play after reaching the target.

xTarget	The x position of the target.
yTarget	The y position of the target.
zTarget	The z position of the target.
tolerance	The distance that an actor should be from the target before it is considered to have reached the target.
Speed	The speed at which the actor should move toward the target.
turnSpeed	The speed at which the actor should turn toward the target.
moveMode	MoveTo – The behavior will run continuously causing the actor to move toward the target even after it moves. MoveToAndStop – The behavior will stop after it successfully reaches the target.

AnimationManager

Plays a specified animation sequence on an actor with an animation manager.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`manager`	The animation manager to use.
`animation`	The animation sequence that the actor should play.
`frequency`	The speed of the animation sequence.
`phase`	The phase of the animation sequence.
`Accumulate`	True – The actor will accumulate any movement that occurs during an animation sequence. False – The actor will start from the animation sequence's original position every time the animation is run.
`Backwards`	True if the animation sequence should be played backwards.
`EndMode`	Stop Anim – The animation sequence will stop when the behavior stops. Continue – The animation sequence will continue even after the behavior stops.
`Looping`	On if the animation sequence should repeat itself when finished.
`Layering`	On if this animation sequence should be blended on top of another animation sequence. There should be at least one animation manager running with layering off. Any additional animation managers may have layering on.

Arithmetic

Calculates the result of an arithmetic operation.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Input 1`	The value of the operation's first operand.
`Operator`	The operator to use – (Add, Subtract, Multiply or Divide)
`Input 2`	The value of the operation's second operand.
`Output`	The result of the operation.
`Run`	Once – The behavior will calculate the result and stop. Continuously – The behavior will continuously calculate the result of the operation as the inputs change.

AttributeValue

Used to set/get the value of a Parameter or Property. This behavior runs once than, stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Mode`	Set – The behavior will set the attribute's value when run. Get – The behavior will get the attribute's value and output it to the 'Value' parameter.
`IdentifierType`	Specifies how the parameter's below will be specified (name or instanceid).
`Type`	Specifies what type of attribute this behavior will set/get. May be an actor property, a joint property or a behavior parameter.
`Actor`	The name/instanceid of the actor that contains the attribute.
`Joint`	The name/instanceid of the joint that contains the property.
`Behavior`	The name /instanceid of the behavior that contains the parameter. If specified by name, a dotted notation such as behaviorName.childbehaviorName.grandchildName… should be used.
`Attribute`	The name/instance id of the attribute that should be used.
`Value`	In set mode, the value to set the attribute to. In get mode, the output parameter for the attribute's value.

AxisRotation

Rotates the actor about an arbitrary axis and center of rotation.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Axis X`	Specifies the x component of the rotational vector's direction.
`Axis Y`	Specifies the y component of the rotational vector's direction.
`Axis Z`	Specifies the z component of the rotational vector's direction.
`axis space`	Global – The axis of rotation is interpreted as vector in world space. Local – The axis of rotation is interpreted as a vector in local space (space of the actor's root).
`COR X`	Specifies the x position of the center that the actor will be rotated about.
`COR Y`	Specifies the y position of the center that the actor will be rotated about.
`COR Z`	Specifies the z position of the center that the actor will be rotated about.
`cor space`	Global – The center of rotation is interpreted as a point in world space. Local – The center of rotation is interpreted as a point in local space (space of the actor's root).
`Angle`	The amount in radians that the actor should rotate.
`Time Unit`	Specifies if the rotated amount should be completed every frame, or every second.
`Run`	Once – The behavior will rotate the actor (ignoring the

	time unit) and stop. Continuously – The behavior will run continuously and will continuously rotate the actor by the amount specified.

CameraProperties

Exposes a subset of the camera's properties. This behavior runs once than stops. This behavior only runs under a camera actor.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`left`	The value of the left extent of the camera's frustum.
`right`	The value of the right extent of the camera's frustum.
`Top`	The value of the top extent of the camera's frustum.
`bottom`	The value of the bottom extent of the camera's frustum.
`front`	The value of the front extent of the camera's frustum.
`back`	The value of the back extent of the camera's frustum.
`fov`	The field of view in radians, this will automatically change the frustum.
`visible`	If false, the geometry for this camera will not be visible. It may still be active even if it is not visible.
`active`	True if the view should look through this camera.

ClampToRange

Clamp’s an input value to be between a range of values.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Value`	The input value to be clamped.
`Min Range`	The Min Range will replace the input value if the input is less than the Min Range.
`Max Range`	The Max Range will replace the input value if the input is grater than the Max Range.
`Output`	The result of the clamp operation.
`Run`	Once – The behavior will calculate the result and stop. Continuously – The behavior will continuously calculate the result of the operation as the inputs and clamp range change.

CombineText

Concatenates two strings of text together.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Text1 In`	The first of two strings to be concatenated.
`Text2 In`	The second of two strings to be concatenated.
`Text Out`	The result of the concatenation.
`Run`	Once – The behavior will calculate the result and stop. Continuously – The behavior will continuously calculate the result of the concatenation as the input text changes.

Convert

Converts a single input (string, float, or int) to a single output (string, float, or int).

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`InputType`	Specifies the type of the first input and which of the corresponding parameters that will be used. (For example, if the input type is Int, the 'Int In' parameter will be used for the conversion.
`Int In`	Used to specify the input as an integer.
`Float In`	Used to specify the input as a float.
`String In`	Used to specify the input as a string.
`OutputType`	Specifies the type that the input should be converted to.
`Int Out`	Conversions with OutputType of Int will be output here.
`Float Out`	Conversions with OutputType of Float will be output here.
`String Out`	Conversions with OutputType of String will be output here.
`Run`	Once – The behavior will calculate the result and stop. Continuously – The behavior will continuously calculate the result of the concatenation as the input changes.

CreateText

Creates a string of text from a formatted string. For every, %s in the formatted string, a new parameter will be created.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Format`	The format string should contain %s place holders any where text is to be created. For example, if the Format is "hello %s", a second Parameter will be created. If the value of that second parameter is "world", the output text will display "hello world".
`OutputText`	The text created from the format string and the input parameters.
`Run`	Once – The behavior will create the text and stop. Continuously – The behavior will continuously create the text as the Format and input parameters change.

CubeMap

Renders a cubemap onto the containing actor's target joint.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Target`	The joint that the cubemap will be applied to.
`Reflect`	Yes – If the cube map should be a rendered texture. No – If the cube map should use example images as the textures.

Cursor

Creates a cursor with a standard or custom image over the viewport. The cursor is available for as long as the behavior is running.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Type`	Specifies the type of image to be used for the cursor.
`CustomImage`	If the Type is set to 'Custom', Custom Image will be the name of the resource to use. This resource must exist in the path.
`HotSpotX`	The x-coordinate of the cursor's hotspot.
`HotSpotY`	The y-coordinate of the cursor's hotspot.
`BoundLeft`	The cursor's hotspot is not allowed beyond this pixel amount from the left of the screen.
`BoundTop`	The cursor's hotspot is not allowed beyond this pixel amount from the top of the screen.
`BoundRight`	The cursor's hotspot is not allowed beyond this pixel amount from the right of the screen.
`BoundBottom`	The cursor's hotspot is not allowed beyond this pixel amount from the bottom of the screen.
`Windowed`	Specifies if the cursor is in a windowed environment. (Not supported.)
`Animated`	Specifies if the cursor is animated. (Not supported.)
`Immediate`	Specifies if the cursor is to be updated immediately instead of per frame. (Not supported.)
`Enabled`	Specifies if the cursor is visible.

DynamicTexture

Attaches an example of a dynamic texture to a joint.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Target`	Specifies the target joint that the Dynamic texture is to be applied on.

FullScreen

Allows switching the view between full screen and windowed mode. This behavior runs once and stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Full Screen`	Yes - if turning full screen on. No - if turning full screen off.
`Adapter`	A list of adapters currently available on this computer.
`Resolution`	A list of resolutions currently available on the chose adapter.

LightProperties

Exposes a subset of the light's properties. This behavior runs once than stops. This behavior only runs under a light actor.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`type`	Specifies the type of light this light actor should switch to.
`r`	Specifies the red component of the lights color. (values can range from 0 to 1)
`g`	Specifies the green component of the lights color. (values can range from 0 to 1)
`b`	Specifies the blue component of the lights color. (values can range from 0 to 1)
`range`	Specifies how far the affect of the light reaches. (Not supported, use attenuation.)
`const atten.`	Specifies the constant attenuation factor.
`lin atten.`	Specifies the linear attenuation factor.
`quad atten.`	Specifies the quadratic attenuation factor.
`penumbra`	For spot lights, this is the angle of space the spotlight affects.
`umbra`	The portion of the penumbra where the penumbra is most intense. (Not supported.)
`visible`	Specifies if this light is visible in the scene, it may be enabled without being visible.
`enabled`	Specifies if the light is on or off.

LockToViewport

Locks a specified joint to the screen so it is always up against the camera.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`x view`	The normalized x screen coordinate of where to place the anchored position anchored at the anchor point. (values range from 0 to 1)
`y view`	The normalized y screen coordinate of where to place the anchored position anchored at the anchor point. (values range from 0 to 1)
`x anchor`	A normalized x coordinate that specifies the point on the object to be locked to the view point. (values range from 0 to 1)
`y anchor`	A normalized y coordinate that specifies the point on the object to be locked to the view point. (values range from 0 to 1)
`NearFactor`	How far away the oject should be from the camera. (a value of 1 puts it at the camera's position, greater values move it farther away.
`camera`	Specifies which camera the joint should be locked to.
`joint`	The joint that will be locked to the viewport.

MouseMonitor

Outputs positional, velocity, and acceleration information of the mouse in screen space.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`x pos`	The x position the mouse, where 0 is the left of the viewport.
`y pos`	The y position of the mouse, where 0 is the top of the view.
`x delta`	The amount of the last horizontal positional change of the mouse.
`y delta`	The amount of the last vertical positional change of the mouse.
`x vel`	The value of the last horizontal velocity (measured per unit time) of the mouse.
`y vel`	The value of the last vertical velocity (measured per unit time) of the mouse.
`x acc`	The value of the last horizontal acceleration (measured per unit time) of the mouse.
`y acc`	The value of the last vertical acceleration (measured per unit time) of the mouse.

NetworkActor

Puts an actor on the network, so that it's attributes can be shared/viewed by others on the network. Note, this does not network the containing actor's properties; only the NetworkActor behavior's parameters are networked.

The table below describes the functionality of each parameter. Both sides of the network should have this behavior under an actor with the same name.

<Parameter>	<Description>
`Class`	The type of object to be placed on the network. The choices for this will be specified by the Network Connection behavior.
`Shared`	True -- Specifies that both sides of the network can change an attribute of this NetworkActor. False – Specifies that the side of the network that runs the behavior first 'owns' the attributes. Only this side may modify them.

NetworkConnection

Connects this application to the network.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Connection Name`	The name of the network connection to connect to.
`App Name`	The name that this application will have on the network.
`Spec File`	The specification file for this connection. If empty, the default spec file is the same as the connection name.
`Class Library`	The name of the class library dll to be loaded. The dll must be in the path.

NetworkInteraction

Sends/Recieves an interaction over the network.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Class`	The type of interaction to send. The choices for this will be specified by the Network Connection behavior.
`TransportMode`	Send – Sends this interaction over the network, then stops. Receive & Continue – The behavior continuously runs and updates its parameters as new interactions of this type are received. Receive & Stop – The behavior stops running when it receives an interaction of this type.
`Filter`	The parameter specified will filter all interactions of this type, so that only interactions with the same value in the filter's parameter are received.

ReflectiveSurface

Reflects the environment as a rendered texture onto the target joint.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Target`	Specifies the target joint of the containing actor that should reflect the environment.
`Reverse Cam`	Specifies if camera should be reversed.
`Flip Horiz`	Flips the image horizontally.
`Flip Vert`	Flips the image vertically.

ResetActor

This behavior will reset the joints on the containing actor. This behavior runs once then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`resetMode`	Default – The joints on the actor will be reset with the roots position and rotation set to (0,0,0) and size set to (1,1,1) Last Saved State – The joints on the actor will be set to their last saved state.
`Save state?`	Yes – Save the current state of the joints, overwriting the previous saved state. No – Reset the joints to either their defaults or last saved state, as specified by the resetMode parameter.

ResetJoint

This behavior will reset a single joint on the containing actor. This behavior runs once then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`joint`	The joint to either reset or save.
`resetMode`	Default – The specified joint will be reset relatively. If it is the root, the position and rotation are set to (0,0,0) and the is size set to (1,1,1) Last Saved State – The specified joint will be set to their last saved state.
`save state?`	Yes – Save the current state of the joint, overwriting the previous saved state. No – Reset the joint to either its defaults or last saved state, as specified by the resetMode parameter.

Rotation

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Xa`	The x component of the rotation (in radians).
`Ya`	The y component of the rotation (in radians).
`Za`	The z component of the rotation (in radians).
`Space`	Global – The rotation is relative to the origin of the world. Local – The rotation is relative to the origin of the Actor
`Apply`	Absolute – The rotation is in world space. Relative - The rotation is relative to the Actor's current rotation.
`Time Unit`	Per Frame – The Actor is rotated by the specified amount every frame (frame rate dependent) Per Second – The Actor is rotated smoothly such that it will have been rotated by the specified amount once every second.
`Run`	Once – Specifies to apply the rotation for one frame only and stop. Continuously – Specifies to continually apply the rotation.

Route

Creates or destroys a route dynamically. This behavior runs once then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Mode`	Create Route – creates a route specified by the parameters below. (If it doesn't already exist.) Destroy Route – destroys a route specified by the parameters below. (If it already exists.)
`Route Type`	This parameter is only necessary when creating a route. ByVal – The route to be created will route from the source by value. ByRef – The route to be created will route from the source to the target by reference, copying the value back to the source. ByValContinuous – The route to be created will continuously route by value from the source to the target.
`Source Type`	Specifies whether the route to be created/destroyed has its source on an actor property, joint property, or behavior parameter.
`Source Actor`	The name of the source actor that the route will be created on/destroyed from.
`Source Joint`	The name of the source joint that the route will be created on from/destroyed from.
`Source Behavior`	The name of the source behavior that the route will be created on/destroyed from. The name should be a dotted notation such as behaviorName.childbehaviorName.grandchildName…
`Source Variant`	The name of the source variant that the created route will route from. Or, the name of the source variant that will identify the route to be destroyed.

`Target Actor`	The name of the target actor that the route will be created to/destroyed from.
`Target Behavior`	The name of the target behavior that the route will be created to/destroyed from. The name should be a dotted notation such as behaviorName.childbehaviorName.grandchildName…
`Target Variant`	The name of the target variant that the created route will route to. Or, the name of the target variant that will identify the route to be destroyed.

SetActiveScene

Specifies the scene to make active. This behavior runs once, then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Scene`	The name of the scene to be activated. This will deactivate the current scene if it is different.

SetVisibility

Sets an actor's visibility flag on or off. This behavior runs once, then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`visible`	Sets the visibility of the containing actor.

SetColorAlpha

Sets the color and/or alpha values on all joints of the containing Actor. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`set color`	Yes – If the specified color should be applied to this actor.
`red`	The red component of the color. (values range from 0 to 1)
`green`	The green component of the color. (values range from 0 to 1)
`blue`	The blue component of the color. (values range from 0 to 1)
`set alpha`	Yes – If the specified alpha value should be applied to this actor.
`alpha`	The alpha value to be applied. (values range from 0 to 1)

SetJointColorAlpha

Sets the color and alpha values of the specified Joint.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`set mode`	Restore Original – The original values will be restored to the Joint (all other parameters are ignored) User Defined – The Joint's color and alpha values are set to the specified values.
`joint`	The Joint to adjust.
`red`	The red component of the color.
`green`	The green component of the color.
`blue`	The blue component of the color.
`alpha`	The alpha component of the color.
`set color`	If Yes, the color values are applied.
`set alpha`	If Yes, the alpha value is applied.

SetJointCOR

Sets the center of rotation that the Joint will use.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`joint`	The joint to set the COR on.
`CoR x`	The X component of the center of rotation.
`CoR y`	The Y component of the center of rotation.
`CoR z`	The Z component of the center of rotation.
`coordSpace`	Global – The COR is specified relative to the world origin. Local – The COR is specified in the Joint's coordinate space.

SetJointVisibility

Shows or hides a Joint of the current Actor.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`joint`	The Joint to show or hide.
`visible`	The Joint is made visible if True, or invisible if False.

SetProperty

Sets the value of a specified property of the current Actor or one of its Joints.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Target`	Specifies properties of a Joint of the Actor or properties of the Actor if <Entire Actor> is chosen.
`Property`	The property to change.
`Value`	The value to set the property to. The name and the type of this parameter change to match the chosen Property.

Shadow

Casts a shadow of the current actor onto the specified Actor or Joint. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Actor`	The Actor to cast the shadow onto.
`Joint`	The Joint of the actor to cast the shadow onto or <Entire Actor> to apply to all of the Joints.
`Detail`	The size of the texture that is generated to create the shadow. It has a width and height of 2^Detail.
`Light Ref`	The name of an Actor to use as a source point of the virtual light that casts the shadow.
`Cast Mode`	Auto – Determines whether to use Parallel or Perspective modes based on the Light type of the Light Ref. If the Light Ref is not specified, the virtual light is position relative to the current Actor and Parallel mode is chosen. Parallel – The shadow is cast onto the target using parallel light rays (shadow is always a fixed size and doesn't skew). Perspective – The shadow is cast onto the target and is skewed and expanded to account for perspective.
`Red`	The red component of the shadow color.
`Green`	The green component of the shadow color.
`Blue`	The blue component of the shadow color.
`Alpha`	The alpha value of the shadow color.
`Enabled`	The shadow is displayed only if Enabled is True

Shape2D

Creates a simple rectangular geometry that is drawn on the view plane of the current camera.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Pos Units`	Normalized – position and anchor values are in normalized screen coordinates (0.0 to 1.0) Pixels – position and anchor values are in pixels (0,0 is the top left of the screen)
`X Pos`	The horizontal position on the screen to place the anchor.
`Y Pos`	The veritical position on the screen to place the anchor.
`X Anchor`	The horizontal position on the rectangle where it is anchored.
`Y Anchor`	The vertical position on the rectangle where it is anchored.
`z depth`	Used to sort the shapes on the screen. Higher values are placed behind lower values.
`Size Units`	Normalized – size values are in normalized screen coordinates (0.0 to 1.0) Pixels – size values are in pixels.
`width`	The width of the rectangle.
`height`	The height of the rectangle.
`texture`	Name of a texture resource to apply to the rectangle.
`red`	The red component of the material color of the rectangle. (0.0 to 1.0)
`green`	The green component of the material color of the rectangle. (0.0 to 1.0)
`blue`	The blue component of the material color of the rectangle. (0.0 to 1.0)

`alpha`	The alpha component of the material of the rectangle. (0.0 to 1.0)
`Visible`	The rectangle is drawn only if visible is True.

SubJointCameraProperties

Sets a subset of the properties of a camera that is attached to the hierarchy of a Joint's model.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Joint`	The Joint that this applies to.
`Camera`	The Camera in the Joint's model hierarchy to adjust.
`Active`	The specified Camera is activated or deactivated.
`Left`	The value of the left extent of the camera's frustum.
`Right`	The value of the right extent of the camera's frustum.
`Top`	The value of the top extent of the camera's frustum.
`Bottom`	The value of the bottom extent of the camera's frustum.
`Front`	The front clip plane distance.
`Back`	The back clip plane distance.
`Fov`	The field of view in radians, this will automatically change the frustum.

SubJointRotation

Rotates a sub component of a model's hierarchy.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Joint`	The Joint that this applies to.
`Node`	The Node in the Joint's model hierarchy to rotate.
`Xa`	Specifies the X angle component of the euler rotation in radians.
`Ya`	Specifies the Y angle component of the euler rotation in radians.
`Za`	Specifies the Z angle component of the euler rotation in radians.
`Space`	Global – The rotation is relative to the origin of the world. Local – The rotation is relative to the origin of the Node
`Apply`	Absolute – The rotation is in world space. Relative - The rotation is relative to the Node's current rotation..
`Time Unit`	Per Frame – The Node is rotated by the specified amount every frame (frame rate dependent) Per Second – The Node is rotated smoothly such that it will have been rotated by the specified amount once every second.
`Run`	Once – Specifies to apply the rotation for one frame only and stop. Continuously – Specifies to continually apply the rotation.

SubJointVisible

Hides or shows a sub component of a model's hierarchy.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Joint`	The Joint that this applies to.
`Node`	The Node in the Joint's model hierarchy to adjust visibility on.
`NodeName`	Duplicate of Node parameter for routing convenience.
`Set Visible`	Specifies if the Node should be made visible or hidden.

Text2D

Displays the specified text on the screen. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`red`	The normalized Red component value of the text color (0.0 to 1.0).
`green`	The normalized Green component value of the text color (0.0 to 1.0).
`blue`	The normalized Blue component value of the text color (0.0 to 1.0).
`alpha`	The normalized Alpha component value of the text color (0.0 to 1.0).
`font`	Specifies the name of the font resource to use.
`text`	The text to be displayed
`X Align`	The horizontal location of the text's anchor.
`Y Align`	The vertical location of the text's anchor.
`X Anchor`	The horizontal location on the text used for alignment.
`Y Anchor`	The vertical location on the text used for alignment.
`x offset`	The horizontal offset (in pixels) to move the text from its alignment position.
`y offset`	The vertical offset (in pixels) to move the text from its alignment position.
`max width`	The maximum number of characters allowed on a line before automatically starting a new line.
`Wrap By`	Character – The text is wrapped to the next line regardless of sentence structure. Word – The text is wrapped to the next line without splitting up any words.
`All Cams`	If True, the text is displayed no matter what camera is

	active, otherwise it is displayed only for the current Camera.
`Visible`	Specifies if the text should currently be displayed.

Transform

Transforms the Actor using a combination of translation, rotation, and scale.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`translation`	Specifies whether translational values are applied (On) or not (Off).
`Xt`	The X component of the translation vector.
`Yt`	The Y component of the translation vector.
`Zt`	The Z component of the translation vector.
`rotation`	Specifies whether rotational values are applied (On) or not (Off).
`Xa`	Specifies the X angle component of the euler rotation in radians.
`Ya`	Specifies the Y angle component of the euler rotation in radians.
`Za`	Specifies the Z angle component of the euler rotation in radians.
`overrideCOR`	Specifies if the CoR values should be used instead of the Actor's own center of rotation.
`CoR x`	The X position of the center of rotation of the Actor.
`CoR y`	The Y position of the center of rotation of the Actor.
`CoR z`	The Z position of the center of rotation of the Actor.
`pinCOR`	If True, the center of rotation is set at the beginning of the transform operation and not updated.
`scaling`	Specifies whether scaling values are applied (On) or not (Off).
`Xs`	The X component of the scale.
`Ys`	The Y component of the scale.

`Zs`	The Z component of the scale.
`moveMode`	SnapToTargetAndStop – Apply the transform values to the Actor immediately and stop the behavior. SnapToTarget – Continually apply the transform values to the Actor. StraightToTargetAndStop - Apply the transform values to the Actor at a constant rate over a period of time equal to moveTime and then stop the behavior. StraightToTarget – Continually apply the transform values to the Actor over a period of time equal to moveTime. Starts up again when the target transform changes. SmoothToTargetAndStop - Apply the transform values to the Actor over a period of time equal to moveTime and then stop the behavior. The velocity starts at zero, accelerates to the midpoint, then decelerates to the final transform values. SmoothToTarget - Continually apply the transform values to the Actor over a period of time equal to moveTime. The velocity starts at zero, accelerates to the midpoint, then decelerates to the final transform values. Starts up again when the target transform changes.
`moveTime`	How long the transform takes to complete.
`abs/rel`	Absolute – The transformation target is in world space. Relative - The transformation target is relative to the Actor's current transformation..
`coordSpace`	Global – The transformation is relative to the origin of the world. Local – The transformation is relative to the origin of the Actor.
`setLocal`	If True, the local coordinate system is recalculated each time this behavior runs.
`focus joint`	The Joint relative to which this transform applies.
`layering`	Not implemented
`layerMode`	Not implemented

`reset?`	If yes, the transform values are reset to identity and applied.

TransformJoint

Transforms the Joint using a combination of translation, rotation, and scale.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`joint`	The Joint for which this transform will apply.
`translation`	Specifies whether translational values are applied (On) or not (Off).
`Xt`	The X component of the translation vector.
`Yt`	The Y component of the translation vector.
`Zt`	The Z component of the translation vector.
`rotation`	Specifies whether rotational values are applied (On) or not (Off).
`Xa`	Specifies the X angle component of the euler rotation in radians.
`Ya`	Specifies the Y angle component of the euler rotation in radians.
`Za`	Specifies the Z angle component of the euler rotation in radians.
`overrideCOR`	Specifies if the CoR values should be used instead of the Joint's own center of rotation.
`CoR x`	The X position of the center of rotation of the Joint.
`CoR y`	The Y position of the center of rotation of the Joint.
`CoR z`	The Z position of the center of rotation of the Joint.
`pinCOR`	If True, the center of rotation is set at the beginning of the transform operation and not updated.
`scaling`	Specifies whether scaling values are applied (On) or not (Off).
`Xs`	The X component of the scale.

`Ys`	The Y component of the scale.
`Zs`	The Z component of the scale.
`moveMode`	SnapToTargetAndStop – Apply the transform values to the Joint immediately and stop the behavior. SnapToTarget – Continually apply the transform values to the Joint. StraightToTargetAndStop - Apply the transform values to the Joint at a constant rate over a period of time equal to moveTime and then stop the behavior. StraightToTarget – Continually apply the transform values to the Joint over a period of time equal to moveTime. Starts up again when the target transform changes. SmoothToTargetAndStop - Apply the transform values to the Joint over a period of time equal to moveTime and then stop the behavior. The velocity starts at zero, accelerates to the midpoint, then decelerates to the final transform values. SmoothToTarget - Continually apply the transform values to the Joint over a period of time equal to moveTime. The velocity starts at zero, accelerates to the midpoint, then decelerates to the final transform values. Starts up again when the target transform changes.
`moveTime`	How long the transform takes to complete.
`abs/rel`	Absolute – The transformation target is in world space. Relative - The transformation target is relative to the Joint's current transformation..
`coordSpace`	Global – The transformation is relative to the origin of the world. Local – The transformation is relative to the origin of the Joint.
`recenterJoint`	If True, the Joint is translated such that its center of geometry is at the specified location.
`layering`	Not implemented
`layerMode`	Not implemented

`reset?`	If yes, the transform values are reset to identity and applied.

Translation

Moves the Actor to a new position.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`X`	Specifies the X component of the translation vector.
`Y`	Specifies the Y component of the translation vector.
`Z`	Specifies the Z component of the translation vector.
`Type`	Direction – The translation vector is interpreted as a unit direction vector and is combined with the Distance parameter to create a velocity vector. Velocity – The translation vector is interpreted as a velocity vector. Its length is the distance value and the Distance parameter is ignored.
`Distance`	Specifies the distance to translate when using a direction vector.
`Space`	Global – The translation direction is relative to the origin of the world. Local – The translation direction is relative to the origin of the Actor.
`Apply`	Relative – The Actor is translated relative to its current position. Absolute – The Actor is translated to the position specified by the velocity vector (which is rotated into the Space specified).
`Time Unit`	Per Frame – The Actor is translated the full length of the velocity vector every frame (frame rate dependent) Per Second – The Actor is translated smoothly in the direction of the velocity vector such that it will have been translated by the full length of the velocity vector once every second.
`Run`	Once – Specifies to apply the translation for one frame only and stop. Continuously – Specifies to continually apply the

	transform.

UpdateAnimations

Updates any animations included in the Actor. Use this if the Actor includes animations that activate automatically and can not be controlled by an AnimationManager. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
Joint	The name of the Joint to update or choose <Entire Actor> to update all of them.

VisualSettings

Changes the visual aspects of a model. This behavior runs for one frame.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
Target	The name of the Joint that will have its visual settings changed.
VisualSetting	The settings category this Behavior will affect -- MatColor, MatAlpha, MatVertCol, MatSpecular, MatShading, MatWireframe, MatDither, MatStencil, ZBuffer, or Fog.
…	The rest of the parameters vary depending on the visual setting category chosen. These parameters allow you to change the various aspects of that category.

ActorKeyMoveAction

The behavior monitors the keyboard and pushes into the output parameter the mapped action and velocity in the X and Z directions.

<Parameter>	<Description>
`IdleAction`	The animation to play when the actor is idle.
`ForwardInput`	The key for moving a character forward.
`ForwardAction`	The animation to play in response to the forward input.
`ForwardVel`	The velocity of the actor when the forward input is received.
`BackwardInput`	The key for moving a character backward.
`BackwardAction`	The animation to play in response to the backward input.

`BackwardVel`	The velocity of the actor when the backward input is received.
`LeftInput`	The key for moving a character left.
`LeftAction`	The animation to play in response to the left input.
`LeftVel`	The velocity of the actor when the left input is received.
`RightInput`	The key for moving a character right.
`RightAction`	The animation to play in response to the right input.
`RightVel`	The velocity of the actor when the right input is received.
`ForwardLeftInput`	The key for moving a character forward and left.
`ForwardLeftAction`	The animation to play in response to the forward and left input.
`ForwardLeftXVel`	The X velocity of the actor when the forward and left input is received.
`ForwardLeftZVel`	The Z velocity of the actor when the forward and left input is received.
`ForwardRightInput`	The key for moving a character forward and right.
`ForwardRightAction`	The animation to play in response to the forward and right input.
`ForwardRightXVel`	The X velocity of the actor when the forward and right input is received.
`ForwardRightZVel`	The Z velocity of the actor when the forward and right input is received.
`BackwardLeftInput`	The key for moving a character backward and left.
`BackwardLeftAction`	The animation to play in response to the backward and left input.
`BackwardLeftXVel`	The X velocity of the actor when the backward and left input is received.
`BackwardLeftZVel`	The Z velocity of the actor when the backward and

	left input is received.
`BackwardRightInput`	The key for moving a character backward and right.
`BackwardRightAction`	The animation to play in response to the backward and right input.
`BackwardRightXVel`	The X velocity of the actor when the backward and right input is received.
`BackwardRightZVel`	The Z velocity of the actor when the backward and right input is received.
`CombineCardinalInputs`	Enables/Disables the combination of inputs.
`CombineCardinalVelocities`	Enables/Disables the combination of velocities.
`CurrentAction`	The action corresponding to the currently received input.
`CurrentXVel`	The current X velocity based on the currently received input.
`CurrentZVel`	The current Z velocity based on the currently received input.

ArithmeticEx

This behavior calculates the result of an arithmetic operation.

<Parameter>	<Description>
`IdentifierType`	The type of identifier to use for locating an attribute.
`Input1Type`	The type of the operation's first operand.
`Input1Value`	The value of the operation's first operand.
`Operator`	The operator to use – (Add, Subtract, Multiply or Divide)
`Input2Type`	The type of the operation's second operand.
`Input2Value`	The value of the operation's second operand.
`OutputType`	The type of the result of the operation.
`OutputValue`	The result of the operation.

CollectionValue

This behavior manages the values of a collection type parameter or property.

<Parameter>	<Description>
`CollectionOwnerType`	The type of the collection’s owner.
`CollectionLocator.Scene`	The scene specifier of the target collection’s locator.
`CollectionLocator.Actor`	The actor specifier of the target collection’s locator.
`CollectionLocator.Attribute`	The attribute specifier of the target collection’s locator.
`Function`	The function to perform on the collection. Available operations are Insert, Remove, Set Value, Get Value.
`Mode`	Determines how the Function will be applied to the collection.
`ElementLocator`	The locator of the element to which you wish to manage.
`ValueName`	The name of the attribute to manager.
`Value`	The value of the attribute being managed.

CollisionRadius

The Collision Radius behavior detects collision between specified targets at a specified distance.

<Parameter>	<Description>
`Targets`	Collection of actors to measure
`Distance`	Offset from target used to determine a collision.
`IsColliding`	Output parameter used to determine when a collision is detected. Parameter is True when a collision is detected.

HandleCollisionsEx

The HandleCollisionsEx behavior pushes back its containing actor to the last position where it was not colliding with an actor on the Target list.

<Parameter>	<Description>
`InputPosition`	Tuple indicating current position of owning actor.
`Targets`	Collection of actors to monitor collisions with.
`TargetsAreMobile`	Hint to system as to whether any of the Targets can move.
`RepelSpeed`	Speed with which the containing actor is pushed away from a target.
`RepelMinOffset`	Minimum offset from which the containing actor is pushed away from a target.
`RepelFilter`	Removes X, Y, or Z from being repelled.
`OwnerBoundType`	Containing Actor’s bounding volume.
`TargetBoundType`	Target Actor’s bounding volume.

`IsColliding`	Output parameter indicating whether the containing actor is colliding with any of the target actors.
`OutputPosition`	Tuple indicating new position of the owning actor.

LineOfSight

This behavior determines whether the source actor is in the line of sight of the target actor.

<Parameter>	<Description>
`SourceActor`	Actor from which to determine if line of sight exists.
`SourceOffset`	Offset from source actor from which to determine if line of sight exits.
`TargetActor`	Actor to which to determine if line of sight exists.
`TargetOffset`	Offset from the target actor from which to determine if line of sight exists.
`Run`	Determines if the behavior runs once or continuously.
`IsInSight`	Boolean output parameter that indicates whether the target actor is in the source actors line of sight.

NetworkActor2

Puts an actor on the network, so that it's attributes can be shared/viewed by others on the network. Note, this does not network the containing actor's properties; only the NetworkActor behavior's parameters are networked.

The table below describes the functionality of each parameter. Both sides of the network should have this behavior under an actor with the same name.

<Parameter>	<Description>
`Class`	The type of object to be placed on the network. The choices for this will be specified by the Network Connection behavior.
`Shared`	True -- Specifies that both sides of the network can change an attribute of this NetworkActor. False – Specifies that the side of the network that runs the behavior first 'owns' the attributes. Only this side may modify them.

NetworkConnection2

The NetworkConnecton2 behavior connects the containing application to the network.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Connection Name`	The name of the network connection to connect to.
`App Name`	The name that this application will have on the network.
`Spec File`	The specification file for this connection. If empty, the default spec file is the same as the connection name.
`Class Library`	The name of the class library dll to be loaded. The dll must be in the path.

NetworkInteraction2

The Network Interaction behavior Sends/Recieves an interaction over the network.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Class`	The type of interaction to send. The choices for this will be specified by the Network Connection behavior.
`TransportMode`	Send – Sends this interaction over the network, then stops. Receive & Continue – The behavior continuously runs and updates its parameters as new interactions of this type are received. Receive & Stop – The behavior stops running when it receives an interaction of this type.
`Filter`	The parameter specified will filter all interactions of this type, so that only interactions with the same value in the filter's parameter are received.

ScreenShader

Renders the screen buffer to a polygon and applies a shader to that polygon. The polygon is then displayed at the specified coordinates.

<Parameter>	<Description>
`X Pos`	X coordinate of the screen polygon.
`Y Pos`	Y coordinate of the screen polygon.
`Z Depth`	Z coordinate of the screen polygon.
`Width`	Normalized width of the screen polygon.
`Height`	Normalized height of the screen polygon.
`Shader Filename`	File name of the shader applied to the screen polygon.
`Shader Name`	Name of the shader applied to the screen polygon.
`Num Shader maps`	Number of shader maps contained in the file.

SoundProperties

This Behavior sets global sound properties on the Audio Adapter.

<Parameter>	<Description>
`Speaker Mode`	Sets the mode for the users speaker setup.
`Pan Seperation`	Sets the master pan seperation for 2d sound effects.
`Distance Factor`	Sets the 3d engine relative distance factor, compared to 1.0 meters. It equates to 'how many units per meter' your 3d engine has.
`Doppler Factor`	Sets the doppler shift scale factor.
`Rolloff Factor`	Sets the global attenuation rolloff factor. Normally volume for a sample will scale at 1 / distance. This gives a logarithmic attenuation of volume as the source gets further away (or closer). Setting this value makes the sound drop off faster or slower. The higher the value, the faster volume will fall off. The lower the value, the slower it will fall off. For example a rolloff factor of 1 will simulate the real world, where as a value of 2 will make sounds attenuate 2 times quicker.
`Master Volume`	Sets the master volume for any sound effects played.
`Run`	Runs the behavior either once or continuously.

Velocity

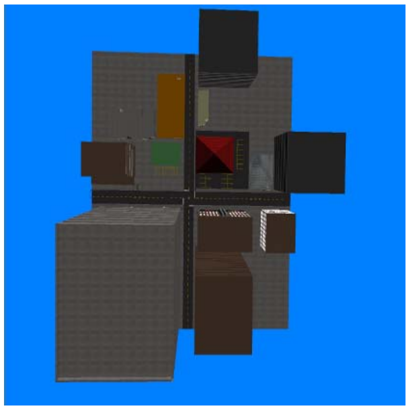
Determines total distance traveled in the current frame.

<Parameter>	<Description>
`Input`	Distance
`Velocity`	Velocity
`Output`	Input+DeltaTime*Velocity

TRAFFIC COMPONENTS

Traffic Components are realized in the TrafficComponents.dll file or in the <Install>.Components\Traffic directory.

City



City

The City Actor consists only of geometry. The City Actor serves as the setting for the Traffic Demo.

<Property>	<Description>
None	

<Animation>	<Description>
None	

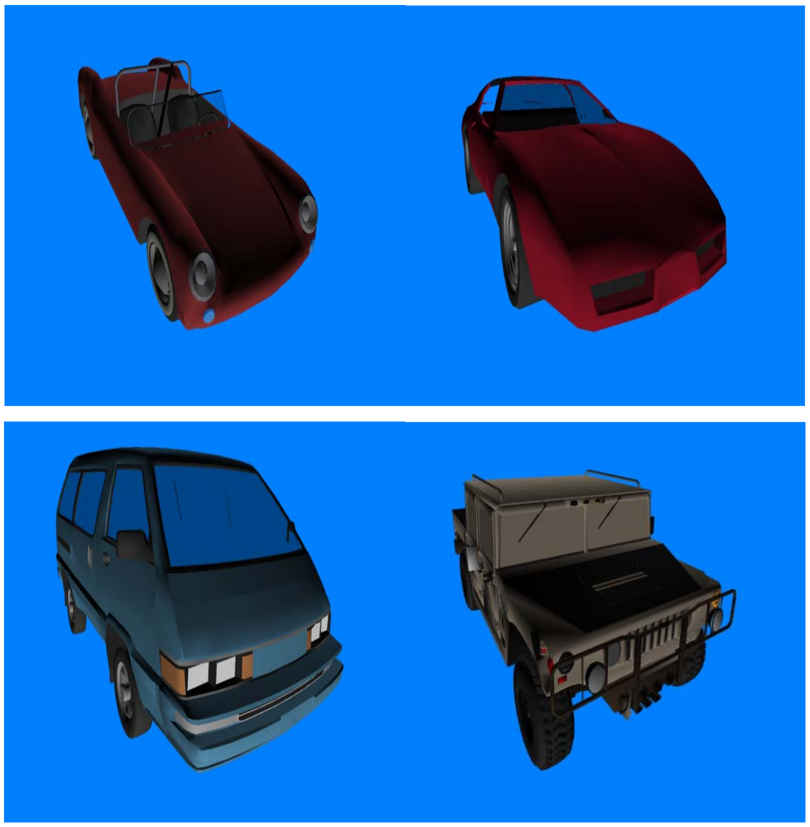
Vehicle

The vehicle actor works in conjunction with the Traffic Components behaviors by allowing the behaviors to set and get information such as the Control Command, the aggressiveness of the driver, various dimensions, position, speed, and acceleration.

<Property>	<Description>
`None`	

<Animation>	<Description>
`None`	

Cars


Cars

The Car Actors consist only of geometry and inherit from Vehicle. Each car represents a different make and model. A New York Taxi, Mazda MX5, Toyota Hilux, VW Coccinelle, Chevrolet Camaro, Suzuki Jimmy, Jeep Cherokee, Porche Speedster 1955, Toyota Land Cruiser, Chevrolet Corvette, Toyota Van, and Hummer are represented.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Collision

The Collision behavior can be used when a collision occurs. This behavior stops all behaviors under the Vehicle actors specified by the 'Choice' parameter.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Choice`	StopAll – Stop all behaviors under any actors of type Vehicle. StopActor – Stop all behaviors under the containing Vehicle actor. Ignore – Do nothing.

ControlPanel

The ControlPanel behavior issues the specified Control command to any Vehicle actor in the specified range and direction. This behavior runs once than stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Control`	Stop, Left, Go, or Right – These control commands are set on any Vehicles within range and facing the direction specified below.
`Center X`	The X coordinate of the center of area in which all vehicles will be issued the control command.
`Center Y`	The Y coordinate of the center of area in which all vehicles will be issued the control command.
`Center Z`	The Z coordinate of the center of area in which all vehicles will be issued the control command.
`Radius`	The distance from the center that defines the area in which all vehicles will be issued the control command.
`Dir. X`	This value must be within 45 degrees of the Rot X property on any Vehicle actor that is to be issued the control command. (in radians)
`Dir. Y`	This value must be within 45 degrees of the Rot Y property on any Vehicle actor that is to be issued the control command. (in radians)
`Dir. Z`	This value must be within 45 degrees of the Rot Z property on any Vehicle actor that is to be issued the control command. (in radians)

Drive

This behavior continuously updates a Vehicle's position based on its calculations for the distance, speed, and acceleration of the Vehicle. This behavior continuously runs until it reaches the specified distance or another vehicle, then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Control`	Distance – The behavior will calculate the end positions and rotations based on the initial position and rotation of the actor and the specified Distance parameter. Position – The behavior will use the specified end positions and initial rotations. PositionAndRotation – The behavior will use the specified end positions and end rotations.
`Distance`	The total distance that the vehicle should be moved.
`XInit Pos`	The initial X position of the Vehicle to be used. (Not Supported)
`YInit Pos`	The initial Y position of the Vehicle to be used. (Not Supported)
`ZInit Pos`	The initial Z position of the Vehicle to be used. (Not Supported)
`XMid Pos`	The middle X position of the Vehicle to be used. (Not Supported)
`YMid Pos`	The middle Y position of the Vehicle to be used. (Not Supported)
`ZMid Pos`	The middle Z position of the Vehicle to be used. (Not Supported)
`XEnd Pos`	The end X position of the Vehicle to be used for Position and PositionAndRotation mode.
`YEnd Pos`	The end Y position of the Vehicle to be used for Position and PositionAndRotation mode.
`ZEnd Pos`	The end Z position of the Vehicle to be used for

	Position and PositionAndRotation mode.
`XInit Rot`	The initial X rotation of the Vehicle to be used for Position mode.
`YInit Rot`	The initial Y rotation of the Vehicle to be used for Position mode.
`ZInit Rot`	The initial Z rotation of the Vehicle to be used for Position mode.
`XMid Rot`	The middle X rotation of the Vehicle to be used. (Not Supported)
`YMid Rot`	The middle Y rotation of the Vehicle to be used. (Not Supported)
`ZMid Rot`	The middle Z rotation of the Vehicle to be used. (Not Supported)
`XEnd Rot`	The end X rotation of the Vehicle to be used for PositionAndRotation mode.
`YEnd Rot`	The end Y rotation of the Vehicle to be used for PositionAndRotation mode.
`ZEnd Rot`	The end Z rotation of the Vehicle to be used for PositionAndRotation mode.
`UseInit. V`	Determines if an initial velocity should be used.
`Init V`	The initial velocity to use.
`UseEnd V`	Determines if the end speed should be used.
`End V`	The speed to use while decelerating.
`Has SpdLmt`	Determines if there is a max speed. (Not Supported.)
`Speed Lmt`	The max speed. (Not Supported.)
`Drive Time`	The total time it takes to reach the end positions. This includes any acceleration, deceleration time.
`Accel. Time`	The total time needed to accelerate.
`Brake Time`	The total time needed to decelerate.

Driver Type	Specifies if the driver is of type Aggressive, Mild, Precaution, or User Defined. If not user defined, this parameter defines the Safety Coef to be 0.5, 0.75, 1.0 respectively.
Safty Coef.	Used if the driver type is user defined. Values range from 0.0 to 1.0. The higher the safety Coef, the farther away a Vehicle will stop behind another Vehicle.

RandomSource

This behavior chooses a source (East, South, West, or North) to move a vehicle to. The source is chosen based on the specified probabilities. The behavior runs until the Vehicle is moved to its source, then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`East Xt`	The vehicle will be at this X position if the East source is chosen.
`East Yt`	The vehicle will be at this Y position if the East source is chosen.
`East Zt`	The vehicle will be at this Z position if the East source is chosen.
`East Xa`	The vehicle will be at this X rotation if the East source is chosen.
`East Ya`	The vehicle will be at this Y rotation if the East source is chosen.
`East Za`	The vehicle will be at this Z rotation if the East source is chosen.
`East Prob.`	The probability that the east source will be chosen.
`South Xt`	The vehicle will be at this X position if the South source is chosen.
`South Yt`	The vehicle will be at this Y position if the South source is chosen.
`South Zt`	The vehicle will be at this Z position if the South source is chosen.
`South Xa`	The vehicle will be at this X rotation if the South source is chosen.
`South Ya`	The vehicle will be at this Y rotation if the South source is chosen.
`South Za`	The vehicle will be at this Z rotation if the South source is chosen.

`South Prob.`	The probability that the South source will be chosen.
`West Xt`	The vehicle will be at this X position if the West source is chosen.
`West Yt`	The vehicle will be at this Y position if the West source is chosen.
`West Zt`	The vehicle will be at this Z position if the West source is chosen.
`West Xa`	The vehicle will be at this X rotation if the West source is chosen.
`West Ya`	The vehicle will be at this Y rotation if the West source is chosen.
`West Za`	The vehicle will be at this Z rotation if the West source is chosen.
`West Prob.`	The probability that the West source will be chosen.
`North Xt`	The vehicle will be at this X position if the North source is chosen.
`North Yt`	The vehicle will be at this Y position if the North source is chosen.
`North Zt`	The vehicle will be at this Z position if the North source is chosen.
`North Xa`	The vehicle will be at this X rotation if the North source is chosen.
`North Ya`	The vehicle will be at this Y rotation if the North source is chosen.
`North Za`	The vehicle will be at this Z rotation if the North source is chosen.
`North Prob.`	The probability that the North source will be chosen.

RandomTurn

The random turn behavior starts by signaling a direction (left, right, or straight) on the vehicle based on the specified probabilities. The behavior continues to run until the vehicle receives a control command. The behavior then executes the control command and stops when finished.

The table below describes the functionality of each parameter.

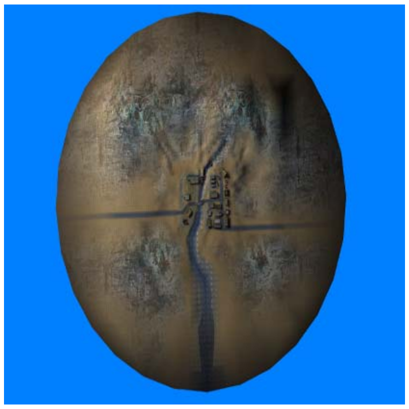
<Parameter>	<Description>
`LTurn Prob.`	The probability that a left turn will be chosen.
`LTurn R`	The radius that defines the arc made by the left turn.
`LTurn A`	The total angle which the vehicle will have turned after smoothly completing the left turn.
`Fwd Prob.`	The probability that a forward direction will be chosen.
`Fwd Dist.`	The total distance to move forward.
`RTurn Prob.`	The probability that a right turn will be chosen.
`RTurn R`	The radius that defines the arc made by the right turn.
`RTurn A`	The total angle which the vehicle will have turned after smoothly completing the right turn.
`UseInitSpeed`	Determines if an initial speed should be used.
`Init Speed`	The initial speed to use.
`UseEndSpeed`	Determines if the end speed should be used.
`End Speed`	The speed to use while decelerating.
`Turn Time`	The total time it takes to complete the turn.
`Accel. Time`	The total time needed to accelerate.
`Decel. Time`	The total time needed to decelerate.
`Blink Delay`	The total amount of time to wait between signals.
`Color R`	The red component of the blink signal color. (values

	range form 0 to 255)
`Color G`	The green component of the blink signal color. (values range form 0 to 255)
`Color B`	The blue component of the blink signal color. (values range form 0 to 255)
`Color A`	The alpha component of the blink signal color. (values range form 0 to 1)

FPS COMPONENTS

FPS Components are realized in the FPSComponents.dll file or in the <Install>.Components\FPS directory.

DesertTown


Desert Town

The DesertTown actor consists only of geometry representing an Iraqi Town. The Desert Town is the setting of the FPS demo.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Insurgent A-E

Insurgent A

The Insurgent Actors consist of geometry and animations. Each Insurgent actor's geometry is slightly different. Variations include skin color, clothing, and weapon.

<Property>	<Description>
None	

<Animation>	<Description>
Crouch	
CrouchBAim	
CrouchBReady	
CrouchFAim	
CrouchFReady	
CrouchLAim	
CrouchLReady	
CrouchRAim	
CrouchReadyToStand	

CrouchReadyToStandReady	
CrouchRReady	
CrouchTurnLAim	
CrouchTurnRAim	
DieB	
DieF	
DieL	
DieR	
Fire	
FireSingleCrouchAim	
FireSingleStandAim	
GestureBeckonAim	
GestureChatReady	
GestureCheerAim	
GestureHaltAim	
GesturePointAim	
GestureTauntAim01	
GestureTauntAim02	
GestureTauntAim03	
HitB	
HitF	
HitR	
IdleCrouchAim	
IdleCrouchReady	
IdleStand	

IdleStandAim	
IdleStandReady	
IdleStandReadyStrech	
RunBAim	
RunF	
RunFAim	
RunFReady	
RunFReadySkidToStop	
RunFReadyTurnLeft	
RunFReadyTurnRight	
RunFSkidToStop	
RunFTurnLeft	
RunJumpBAim	
RunJumpFAim	
RunJumpLAim	
RunJumpRAim	
RunLAim	
RunRAim	
ScanAim	
ScanReady	
StandAimToCrouchAim	
StandAimToCrouchReady	
StandAimToStand	
StandAimToStandReady	
StandReadyToCrouch	

StandReadyToCrouchAim	
StandReadyToStand	
StandReadyToStandAim	
StandToCrouchReady	
StandToStandAim	
StandToStandReady	
TurnLeft	
TurnLeftAim	
TurnLeftReady	
TurnRight	
TurnRightAim	
TurnRightReady	
WalkB	
WalkBAim	
WalkBReady	
WalkF	
WalkFAim	
WalkFReadyWalkLReady	
WalkRAim	
WalkRReady	
WalkTL	
WalkTLAim	
WalkTLReady	
WalkTR	
WalkTRAim	

WalkTRReady	
WalkLAim	

Special Forces A-E

Special Forces A

The Special Forces Actors consist of geometry and animations. Each Special Forces actor's geometry is slightly different. Variations include skin color, clothing, and weapon.

<Property>	<Description>
`None`	

<Animation>	<Description>
`Crouch`	
`CrouchBAim`	
`CrouchBReady`	
`CrouchFAim`	
`CrouchFReady`	
`CrouchLAim`	
`CrouchLReady`	
`CrouchRAim`	
`CrouchReadyToStand`	

CrouchReadyToStandReady	
CrouchRReady	
CrouchTurnLAim	
CrouchTurnRAim	
DieB	
DieF	
DieL	
DieR	
Fire	
FireSingleCrouchAim	
FireSingleStandAim	
GestureBeckonAim	
GestureChatReady	
GestureCheerAim	
GestureHaltAim	
GesturePointAim	
GestureTauntAim01	
GestureTauntAim02	
GestureTauntAim03	
HitB	
HitF	
HitR	
IdleCrouchAim	
IdleCrouchReady	
IdleStand	

IdleStandAim	
IdleStandReady	
IdleStandReadyStrech	
RunBAim	
RunF	
RunFAim	
RunFReady	
RunFReadySkidToStop	
RunFReadyTurnLeft	
RunFReadyTurnRight	
RunFSkidToStop	
RunFTurnLeft	
RunJumpBAim	
RunJumpFAim	
RunJumpLAim	
RunJumpRAim	
RunLAim	
RunRAim	
ScanAim	
ScanReady	
StandAimToCrouchAim	
StandAimToCrouchReady	
StandAimToStand	
StandAimToStandReady	
StandReadyToCrouch	

StandReadyToCrouchAim	
StandReadyToStand	
StandReadyToStandAim	
StandToCrouchReady	
StandToStandAim	
StandToStandReady	
TurnLeft	
TurnLeftAim	
TurnLeftReady	
TurnRight	
TurnRightAim	
TurnRightReady	
WalkB	
WalkBAim	
WalkBReady	
WalkF	
WalkFAim	
WalkFReadyWalkLReady	
WalkRAim	
WalkRReady	
WalkTL	
WalkTLAim	
WalkTLReady	
WalkTR	
WalkTRAim	

WalkTRReady	
WalkLAim	

Hostage

Hostage

The Hostage Actor consists of geometry and animations.

<Property>	<Description>
None	

<Animation>	<Description>
DieB	
DieF	
DieL	
DieR	
HitB	
HitF	
HitL	
HitR	
IdleStand	

RunF	
RunFSkidToStop	
RunFTurnLeft	
TurnLeft	
RunFTurnRight	
TurnRight	
WalkB	
WalkF	
WalkTL	
WalkTR	

AnimateAction

This behavior selects various animations based on the action that the Actor is currently doing. (Ex. The run animation is played when the action is run.) The action itself is determined from the following actor properties: 'IsDead', 'DangerLevel', 'AimTarget', 'MoveTarget', 'IsMoving', 'IsCrouching', and 'IsSearching'. While the 'IsDead' and 'DangerLevel' properties are assumed to already exist, the remaining properties are created if they don't already exist on the Actor. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`IdleBase`	The base idle animation to be played when idle.
`IdleExtra1`	An extra idle animation to be played when idle.
`IdleExtra2`	An extra idle animation to be played when idle.
`IdleReady`	An idle animation to be played when ready.
`IdleAlert`	An idle animation to be played when alert.
`Search`	An animation to be played when searching.
`Walk`	An animation to be played when walking.
`WalkReady`	A walk animation to be played when ready.
`WalkSpeed`	The speed of the above walk animations.
`Run`	An animation to be played when running.
`RunAimForward`	An animation to be played when running forward and aiming.
`RunAimBack`	An animation to be played when running backward and aiming.
`RunAimLeft`	An animation to be played when running left and aiming.
`RunAimRight`	An animation to be played when running right and aiming.

`RunSpeed`	The speed of the above run animations.
`Crouch`	An animation to be played when crouching.
`DistTol`	The distance an actor should be from it's MoveTarget, before it is considered to have reached its target.
`TurnSlowSpeed`	How fast the actor should turn while walking.
`TurnFastSpeed`	How fast the actor should turn while running.
`HitForward`	An animation to be played when being hit in the forward direction. (not supported)
`HitBack`	An animation to be played when being hit in the backward direction. (not supported)
`HitLeft`	An animation to be played when being hit in the left direction. (not supported)
`HitRight`	An animation to be played when being hit in the right direction. (not supported)
`DieForward`	An animation to be played when dieing in the forward direction.
`DieBack`	An animation to be played when dieing in the backward direction.
`DieLeft`	An animation to be played when dieing in the left direction.
`DieRight`	An animation to be played when dieing in the right direction.

AttackEnemy

This behavior causes its containing actor to automatically fire at the enemy with specified error and damage factors. The parameters that are used during a particular fire are determined based on the following actor properties: GroupID, DangerLevel, AimTarget, IsDead, Health, IsFiring, IsMoving, and IsCrouching. Another actor is determined to be an enemy only fired if it has a different GroupID than the containing actor. While the 'GroupID' property is assumed to exist, the other properties are created if they don't already exist on the Actor. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`LowError`	The error factor to use when firing with low error, or firing precisely. (values range from 0.0 to 0.05)
`MediumError`	The error factor to use when firing with medium error. (values range from 0.05 to 0.15)
`HighError`	The error factor to use when firing with high error, or firing badly. (values range from 0.15 to 0.25)
`HeadOffset`	The length of the head offset, used to ensure the fire is on target.
`SightFOV`	The angular field of view that the actor has to determine if an enemy is visible. (in radians)
`MinDamage`	The amount of health the enemy looses when fired at is based on the MinDamage and the MaxDamage parameters, as well as the distance from the enemy.
`MaxDamage`	The amount of health the enemy looses when fired at is based on the MinDamage and the MaxDamage parameters, as well as the distance from the enemy.
`MinDelay`	The minimum amount of time to wait before firing.
`MaxDelay`	The maximum amount of time to wait before firing.

FollowTerrain

The FollowTerrain behavior automatically follows the 'DT_Ground' terrain so that the actor is always at ground level.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Offset`	The distance above the ground level the actor should be.

FPSShoot

This behavior causes its containing actor to fire in a forward direction with specified error factors. The amount of health the target looses is random. This behavior is run once, then stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`LowError`	The error factor to use when firing with low error, or firing precisely. (values range from 0.0 to 0.05) (Not supported)
`MediumError`	The error factor to use when firing with medium error. (values range from 0.05 to 0.15) (Not supported)
`HighError`	The error factor to use when firing with high error, or firing badly. (values range from 0.15 to 0.25) (Not supported)

HandleCollisions

The HandleCollisions behavior pushes back its containing actor to the last position where it was not colliding with an actor that has a “proxy” or the buildings in the FPSDemo. This behavior runs continuously.

This behavior has no parameters.

ScreenMap

The screen map behavior displays an onscreen map (‘briefingmap.bmp’) and an arrow (‘arrow.tga’) that signifies the containing actor’s location on the map. This behavior runs continuously.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`Map Size`	The normalized size of the map relative to the screen. (values range from 0.1, smallest to 1.0, largest)
`Zoom`	The zoom level applied to the map. (values range from 0.1, zoomed in to 1.0, zoomed out)

SenseDanger

The SenseDanger behavior is responsible for setting the containing actor's danger level, which is used by the AttackEnemy and AnimateAction behaviors. The danger level is an actor property that has possible values of 0 (not in danger), 1 (ready), and 2 (alert). The danger level is determined based on the specified parameters and the following Actor properties: 'GroupID', 'DangerLevel', 'AimTarget', 'IsDead', 'Health', 'IsSearching', 'IsFiring', and 'IsMoving'. While the 'IsSearching', 'IsFiring' and 'IsMoving' properties are assumed to exist, the other properties are created on the containing actor if they don't exist already. This behavior runs once than stops.

The table below describes the functionality of each parameter.

<Parameter>	<Description>
`HeadOffset`	The length of the head offset, used to ensure that an enemy is visible or not.
`HearGunRad`	The maximum distance that an actor can hear a gun shot from.
`HearMoveRad`	The maximum distance that an actor can hear a moving enemy from.
`SightDist`	The maximum distance that an actor can see from.
`SightFOV`	The angular field of view that the actor has to determine if an enemy is visible. (in radians)
`ready delay`	The amount of time to wait before switching to a danger level of ready.
`idle delay`	The amount of time to wait before switching to a danger level of idle.

CSTT COMPONENTS

CSTT Components are found in the <Install>.Components\CSTT directory.

Chem Guy

Chem Guy

The Chem Guy Actor is used to represent all members of the CST. The Hazmat Survey Team Leader will be wearing Level "A" Protective Clothing. Level "A" Equipment consists of a positive-pressure, supplied air respirator; fully encapsulating vapor or gas-tight chemical resistant suit; inner clothing, chemical gloves, inner and outer; chemical-resistant safety boots; two-way radio. The three members of team and their responsibilities are:

The HazMat Team Leader's responsibilities are as follows:

- Supervise deployment and operation of all monitoring, detection and sampling equipment
- Direct all movement of the HazMat Survey Team in the hot zone.
- Maintain visual accountability of all team members while in the hot zone at all times.
- Generate team log while down range. Log will include at a minimum:
    - Sector sketches (photo or video)
    - Background / Pre-entry equipment readings
    - Time, Location and type of sample
    - Supervise obtainment of sample
        - Identify sample location
        - Take Pictures before and after sample collection
        - Determine Sample number
    - Be prepared to direct rescue operations in the hot zone.

- ***Sampler***. The Sampler will be wearing the same Level "A" Protective Clothing and a two-way radio as the Team Leader. Sampler responsibilities include:
    - Receive pre/post-entry medical exam.
    - Attend mission briefs and ask pertinent questions to clarify mission.
    - Identify and inform Team Leader of any and all additional equipment needs.
    - Perform all tasks in the hot zone as directed by the Team Leader
    - Obtain samples
        - Decon gloves
        - Identify equipment needed for sample
        - Receive equipment from sampler's assistant
        - Take sample
        - Decon gloves and sample container
        - Apply Parafilm seal
        - Clean up Sample fields

- ***Assistant Sampler***. The Assistant Sampler will be wearing the same Level "A" Protective Clothing and a two-way radio as the Team Leader. Assistant Sampler responsibilities include:
    - Receive pre/post-entry medical exam.
    - Attend mission briefs and ask pertinent questions to clarify mission.
    - Identify and inform Team Leader of any and all additional equipment needs.
    - Perform all tasks in the hot zone as directed by the Team Leader
    - Obtain sample
        - Decon gloves
        - Prepare equipment drop and sample fields
        - Prepare equipment needed for sample
        - Over pack
        - Security Seal
        - Place sample in ziplock bag

Each CST member has with him a collection of equipment that he must utilize to successfully complete the scenario. The following is a list of that equipment.

- ***Two-way Radio.*** Allows team members to communicate with the HazMat Section Officer and incident command.
- ***Level "A" PPE.*** This personal protection equipment (PPE) provides the virtual responder with the highest protection against unknown respiratory, liquid and vapor chemical hazards. The enclosed self-contained breathing apparatus (SCBA) will provide up to one-hour or breathing time for the responder.

- ***Air Meter***. An Air meter will show available air that can be supplied to the suit. The timer will accurately reflect the operating levels of a standard breathing apparatus, which gives approximately 45 minutes of normal breathing time.
- ***Improved Chemical Agent Monitor (ICAM)***. The ICAM identifies nerve and mustard agent contamination on personnel and equipment. The ICAM provides the operator instantaneous feedback of chemical hazard levels and quickly determines the presence of contamination on personnel and equipment. The ICAM is a handheld, individual-operated post-attack device for monitoring chemical agent contamination on personnel and equipment. The monitor detects and discriminates between vapors of nerve and mustard agents.
- ***Multi-RAE***. A Photo Ionization Detector (PID) measures parts per million (PPM) of Volatile Organic Compounds. (VOCs). This Five (5) gas monitor is worn on the wrist of the Survey Team Chief.
- ***Digital Camera***. Used in Survey and Sampling Missions to document the Sample Collection Procedure
- ***Collection Bag***. Survey team personnel use the Mylar bag as the initial container for such samples as protective masks and filter canisters, individual antidote and decon kits, munition fragments, and other items too large to place in a specimen jar. The bag is also used to package sample containers to ensure a vapor barrier in case the container is broken in transit. The bag acts as an initial or secondary vapor barrier to prevent air from leaking inward and toxic material outward.
- ***Sample Cart***. – See Cart Actor for more details.

<Property>	<Description>
None	

<Animation>	<Description>
camera	
camera_return	
CART	
COM_animation	
crouch	
crouch_get_up	
crouch_interact	
hand_wipe	

icam	
idle	
MultiRae	
packing_animation	
packing_end	
packing_start	
slide_left	
slide_right	
Turn_left	
Turn_right	
walk	
walk_with_cart	

VI Guy

VI Guy

The VI Guy Actor consists of the geometry of a HazMat Section Leader and animations. As in the ECS prototype, Scenario 1 will use 3D character models to depict the following roles. The Virtual Instructor will be modeled after the HazMat Section Leader. Students will play the roles of the 3 man survey team consisting of the Team Leader, Sampler and Sampler's Assistant. The Hazmat Section Officer wears an appropriate uniform (e.g. police, fire, etc.) and a two-way radio.

The HazMat Section Officer's responsibilities are as follows:

- Brief Entry Team on mission to include:
- Situation
- Level of Personal Protection Equipment required
- Safety and health risks / secondary devices
- Entry/exit procedures and decontamination locations
- Communications
- Sampling and handling procedures
- Reporting Procedures
- Equipment to be used
- Location of area for dress-out and recovery
- Assign team rotations and priorities of work
- Monitor/record on air and down range time

- Brief and make recommendations on mission developments to operations center (OPCEN) and Incident Commander
- Provides final authorization to HazMat team on all mission changes based on Incident Commander guidance.
- Conducts debriefing of operations with Entry Team
- Maintains communications with operations center (OPCEN)
- Available to perform all operational tasks in the hot zone in support of the mission.

<Property>	<Description>
None	

<Animation>	<Description>
hands_on_hips	
head_Rub	
idle	
point	
read	
talk_test_1	
talk_test_1	
talk_test_1	
walk	

Cart

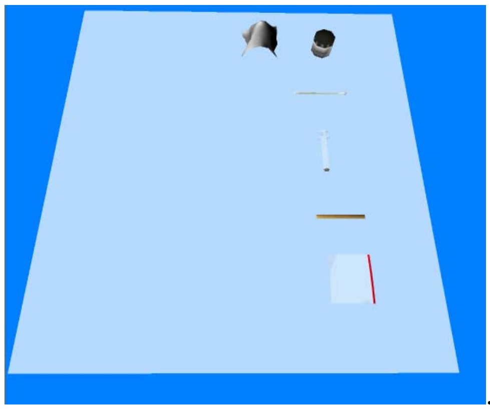
Cart

- The Cart Actor consists only of the geometry of the Cart. This is a cart used by the Survey Team to house miscellaneous, mission critical equipment. The Hapsite is attached on top of this cart Sample Collection kit (M34A1). The M34A1 kit is configured to collect liquid, soil, surface, and small solid samples, suspected of being contaminated with chemical agents, for transport to a laboratory for analysis. This kit consists of miscellaneous sample collection gear including:
    - Bleach Rags
    - Glass Specimen Jars
    - Ruler
    - Parafilm
    - Specimen jar over-packing
    - Security Seal
    - Syringe
    - Swabs
    - Swipes
    - ICAM
    - Multirae
    - Digital Camera

<Property>	<Description>
None	

<Animation>	<Description>
None	

Drop Cloth


.

Drop Cloth

The Drop Cloth Actor consists only of the geometry of the Drop Cloth. This is a plastic sheet used by the Survey Team to layout mission critical equipment without contaminating it.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Dissemination Device

Dissemination Device

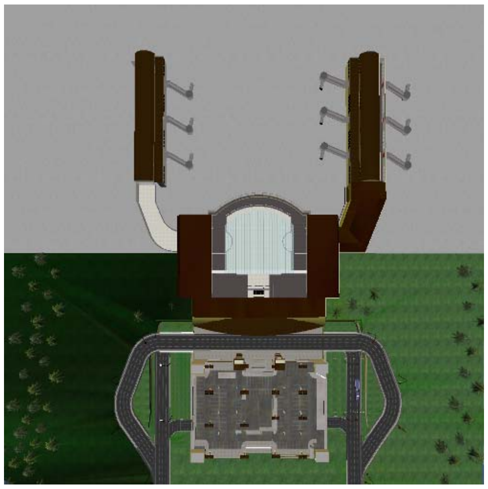
The Dissemination Device Actor consists only of the geometry of the tank. The dissemination device is the focus of the player's attention. It contains an undetermined hazmat that must be properly investigated to successfully complete the scenario.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Airport

The Airport Actor consists only of the geometry of the Airport. The airport environment is the location where the Scenario 1 takes place. The 3D model assets for the airport environment where obtained from the ECS prototype.



Top Down View of the Airport

<Property>	<Description>
None	

<Animation>	<Description>
None	

ERT COMPONENTS

ERT Components are realized in the ERTComponents.dll file or in the <Install>.Components\ERT directory.

Pipe Bomb

Pipebomb

The Pipe Bomb Actor is consists of two parts, the pipe bomb geometry and a particle system that represents the explosion.

<Property>	<Description>
None	

<Animation>	<Description>
Explode	Particle System simulating an explosion

Dynamite

Dynamite

The Dynamite Actor consists of two parts, the dynamite geometry and a particle system that represents the explosion.

<Property>	<Description>
None	

<Animation>	<Description>
Explode	Particle System simulating an explosion

Grenade

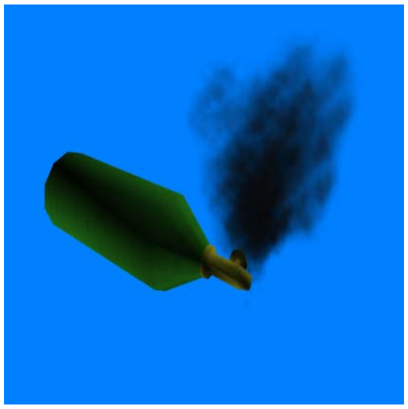
Grenade

The Grenade Actor consists of two parts, the grenade geometry and a particle system that represents the explosion.

<Property>	<Description>
None	

<Animation>	<Description>
Explode	Particle System simulating an explosion

Tank Disseminator



Tank Disseminator

The Tank Disseminator Actor consists of two parts, the Tank geometry and a particle system that represents the substance spraying from it.

<Property>	<Description>
None	

<Animation>	<Description>
Fire	Particle System simulating the release of a burning pressurized substance.

Briefcase Disseminator

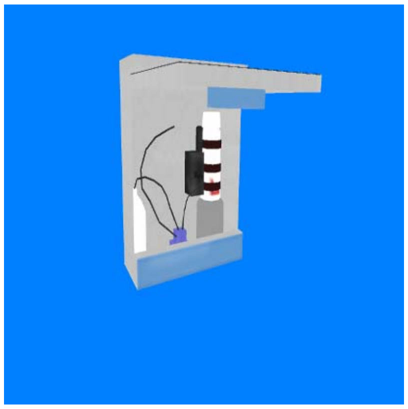
Briefcase Disseminator

The Briefcase Disseminator Actor consists of two parts, the briefcase geometry and a particle system that represents the smoke emanating from it.

<Property>	<Description>
None	

<Animation>	<Description>
Smoke	Particle System simulating smoke.

Powerbox Disseminator



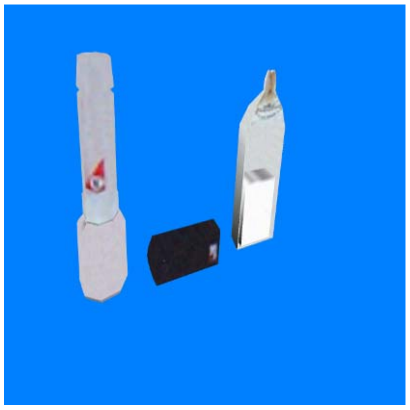
Powerbox Disseminator

The Powerbox Disseminator Actor consists of two parts, the power box geometry and a particle system that represents the smoke emanating from it.

<Property>	<Description>
None	

<Animation>	<Description>
Smoke	Particle System simulating smoke.

Improvised Disseminator


Improvised Disseminator

The Improvised Disseminator Actor consists of two parts, the disseminator geometry and a particle system that represents the smoke emanating from it.

<Property>	<Description>
None	

<Animation>	<Description>
Smoke	Particle System simulating smoke.

Cart

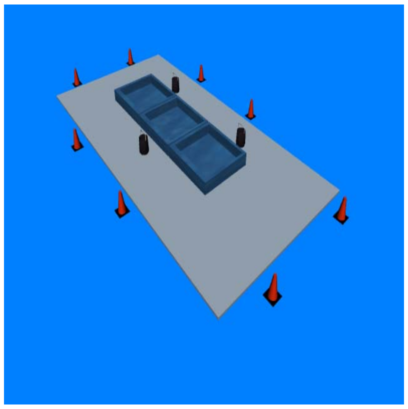
Cart

The Cart Actor consists only of the geometry of the Cart.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Decon


Decon

The Decon Actor consists only of the geometry of the Decon Area. This Actor is used to designate where a team member goes to rehabilitate himself.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Sked

Sked

The Sked Actor consists only of the geometry of the Sked.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Victim01 – 10

Victims 01 - 10

A victim actor consists of geometry and animations. 10 unique victim actors were created for Scenarios 2 and 3. Symptoms and body animations corresponding to exposure to a variety of HazMat/WMD agents were implemented. To properly triage a victim the user must understand and engage the START triage system. Under START, all victims who are able to walk on their own ("walking wounded") are directed to move, or escorted to, a pre-designated decontamination area in the Warm Zone and are labeled as minimal (green tag). This reduces the number of victims to be evaluated. These victims will require supervision and might be detained for further assessment and possible decontamination.

The remaining victims (those who did not follow the direction to relocate to the pre-designated area) will be evaluated using the START triage system. This should take no longer than 1 minute per patient and will focus on three primary areas:

- Respiratory status
- Perfusion and pulse
- Neurological status

As the hazmat medical recon responder moves through each level of assessment, any condition that is deemed immediate (red tag) stops the evaluation process. Life-threatening injuries will be addressed, if necessary, during the initial triage. The patient is tagged, and the responder moves on to the next patient.

Triage Tags

Immediate - Red

When you arrive at an emergency where someone has used the START triage system, your first priority is to find and treat *the* IMMEDIATE patients. These patients are at risk for early death - usually due to shock or a severe head injury. They should be stabilized and transported as soon as possible.

Delayed - Yellow

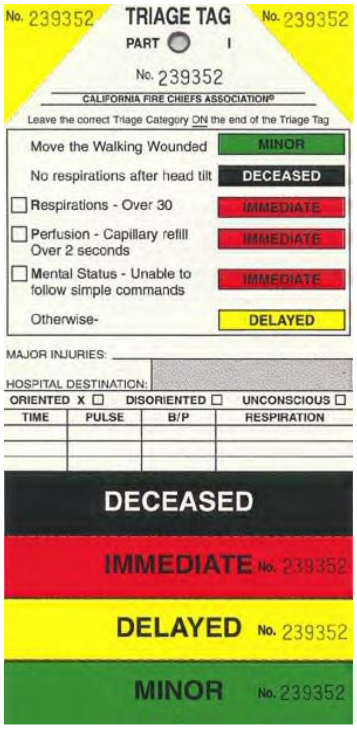
Patients who have been categorized as *DELAYED* are still injured and these injuries may be serious. They were placed in the *DELAYED* category because their respirations were under 30 per minute, capillary refill was under 2 seconds and they could follow simple commands. But they could deteriorate. They should be reassessed when possible and those with the most serious injuries or any who have deteriorated should be top priorities for transport. Also, there may be vast differences between the conditions of these patients. Consider, for example, the difference between a patient with a broken leg and one with multiple internal injuries who is compensating initially. The second patient will need much more frequent re-assessment.

Minor - Green

Patients with *MINOR* injuries are still patients. Some of them may be frightened and in pain. Reassure them as much as you can that they will get help and transport as soon as the more severely injured patients have been transported. Any of these patients also could deteriorate if they had more serious injuries than originally suspected. They should be reassessed when possible.

Deceased - Black

Check with your local protocols about whether patients marked *DECEASED* should be moved. Some systems don't want patients moved until a coroner is on scene, unless they are interfering with rescue attempts.



No. 239352 TRIAGE TAG No. 239352

PART ◯ I

No. 239352

CALIFORNIA FIRE CHIEFS ASSOCIATION©

Leave the correct Triage Category ON the end of the Triage Tag

Move the Walking Wounded	MINOR
No respirations after head tilt	DECEASED
☐ Respirations - Over 30	IMMEDIATE
☐ Perfusion - Capillary refill Over 2 seconds	IMMEDIATE
☐ Mental Status - Unable to follow simple commands	IMMEDIATE
Otherwise-	DELAYED

MAJOR INJURIES:

HOSPITAL DESTINATION:

ORIENTED X ☐ DISORIENTED ☐ UNCONSCIOUS ☐

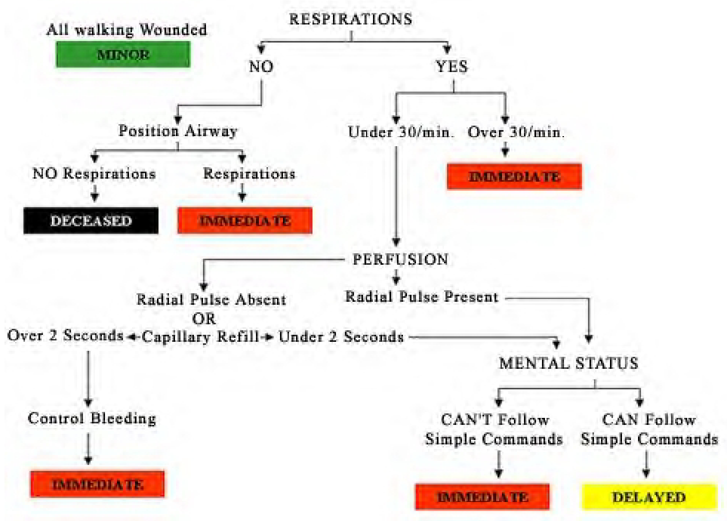
TIME	PULSE	B/P	RESPIRATION

DECEASED

IMMEDIATE No. 239352

DELAYED No. 239352

MINOR No. 239352



Respiratory Status

If the patient is adequately ventilating (breathing), the triage officer moves on to the next step. If, however, ventilation is inadequate, the triage officer attempts to clear the airway by either repositioning the victim or clearing debris from the patient's mouth. If these attempts are unsuccessful, the victim is classified as follows:

- No respiratory effort: expectant (black tag)
- Respirations greater than 30 or needs help maintaining airway: immediate
- Normal respirations: move on to next step.

Signs and Symptoms

The following are signs and symptoms typically used to diagnose a victims state:

EYE PUPILS-N&R: Normal & Reactive

CAR" (Carotid Pulse Present? **Y** (Yes) **N** (No)

RAD: (Radial Pulse Present? **Y** (Yes) **N** (No)

Resp: (Respiration Rate)

Perf: (Perfusion or Blood Circulation Status)

(+2): Capillary Thumbnail refill greater than 2 seconds

(-2): Capillary Thumbnail refill in less than 2 seconds

CAN: (Can understand and follow verbal commands)

CAN'T: (Cannot understand and follow verbal commands)

Victim Cases

The following represent possible victim signs and symptoms the ERT might encounter during their triage mission.

Victim #1

Male patient in his 30's who appears unconscious and is having convulsions. Pupils: Pinpoint, Carotid: Y, Resp: 8, perf: Y(+2), mental :CAN'T, Skin Color: pale. Correct triage selection should be IMMEDIATE (RED). Correct treatment selection should be rapid administration of three-(3) 2mg Atropine Auto injectors, three-(3) 2-PAM Chloride Auto injectors (600 mg each) and one-(1) NAAK autoinjector during the packaging and extraction process. Rapid intervention will result in an improved outcome.

Victim #2

Male patient in his 40's who appears conscious and is displaying dyspnea, blurred vision, throat irritation, and chest tightness. Pupils: Pinpoint, RAD: Y, Resp: 16, perf: Y(-2), mental :CAN, Skin Color: pale. Correct triage selection should be IMMEDIATE (RED). Correct treatment selection should be rapid administration of one-(1) 2mg Atropine Auto injectors, immediately followed by one-(1) 2-PAM Chloride Auto injectors (600 mg each). Rapid intervention will result in an improved outcome.

Victim #3

Female patient in her 50's who appears unconscious and is having convulsions. Pupils: Pinpoint, Carotid: Y, Resp: 8, perf: Y(+2), mental :CAN'T, Skin Color: pale. Correct triage selection should be IMMEDIATE (RED). Correct treatment selection should be rapid administration of three-(3) 2mg Atropine Auto injectors, three-(3) 2-PAM Chloride Auto injectors (600 mg each) and one-(1) NAAK auto injector during the packaging and extraction process. Rapid intervention will result in an improved outcome.

Victim #4

Male patient in his 30's who appears unconscious, possibly seizing or post-dictal, severe twitching or flaccid, dyspnea, vomiting and or defecation and urination These casualties should immediately be given three-(3) 2mg Atropine IM and 2-PAM auto injectors along with one-(1) NAAK (Valium) 10mg auto injector. Pupils: Pinpoint, Carotid: Y, Resp: 8, perf: Y(+2), mental :CAN'T, Skin Color: pale. Correct triage selection should be IMMEDIATE (RED). Rapid intervention will result in an improved outcome.

Victim #5

Male patient in his 20's who appears conscious pinpointed pupils, excess mucous from moth and nose with moderate respiratory distress. Also complains of muscular weakness, vomiting and diarrhea. Initial dose for this patient is 1 MARK I kit containing a total of 2 mg atropine and 600 mg 2-PAM Chloride.

Pupils: Pinpoint, Radial: Y, Resp: 24, perf: Y(+2), mental :CAN, Skin Color: Pink. Correct triage selection should be DELAYED (YELLOW). Rapid intervention will result in an improved outcome.

Victim #5

Female patient in her 60's who appears conscious with complaint of a sudden onset of diminished hearing, lacerations and severe respiratory distress within several minutes following the blast. Pupils: N&R, Radial: Y, Resp: 30 perf: Y(-2), mental :CAN, Skin Color: Pale. Correct triage selection should be IMMEDIATE (RED). Rapid intervention will result in an improved outcome.

Victim #6

Male patient in his 30's who appears conscious with moderate respiratory distress. Also complains diminished hearing. Pupils: N&R, Radial: Y, Resp: 12, perf: Y(-2), mental :CAN, Skin Color: Pink. Correct triage selection should be DELAYED (YELLOW). Rapid intervention will result in an improved outcome.

Victim #7

Male patient in his 40's who appears conscious with minor respiratory distress. Also complains of diminished hearing. Pupils: N&R, Radial: Y, Resp: 12, perf: Y(-2), mental :CAN, Skin Color: Pink. Correct triage selection should be MINOR (GREEN). Rapid intervention will result in an improved outcome.

Victim #8

Female patient in her 50's who appears conscious with pinpointed pupils, excess mucous from moth and nose with moderate respiratory distress. Also complains of muscular weakness, vomiting and diarrhea. Initial dose for this patient is 1 MARK I kit containing a total of 2 mg atropine and 600 mg 2-PAM Chloride.

Pupils: Pinpoint, Radial: Y, Resp: 24, perf: Y(+2), mental :CAN, Skin Color: Pink. Correct triage selection should be DELAYED (YELLOW). Rapid intervention will result in an improved outcome.

Victim #9

Male patient in his 30's who appears unconscious and is having convulsions. Pupils: Pinpoint, Carotid: Y, Resp: 8, perf: Y(+2), mental :CAN'T, Skin Color: pale. Correct triage selection should be IMMEDIATE (RED). Correct treatment selection should be rapid administration of three-(3) 2mg Atropine Auto injectors, three-(3) 2-PAM Chloride Auto injectors (600 mg each) and one-(1) NAAK auto injector during the packaging and extraction process. Rapid intervention will result in an improved outcome.

Victim #10

Male patient in his 40's who appears conscious pinpointed pupils, excess mucous from moth and nose with moderate respiratory distress. Also complains of muscular weakness, vomiting and diarrhea. Initial dose for this patient is 1 MARK I kit containing a total of 2 mg atropine and 600 mg 2-PAM Chloride.

Pupils: Pinpoint, Radial: Y, Resp: 24, perf: Y(+2), mental :CAN, Skin Color: Pink. Correct triage selection should be DELAYED (YELLOW). Rapid intervention will result in an improved outcome.

<Property>	<Description>
None	

<Animation>	<Description>
Choke	
Convulse	
Cough	
Die	
FallDown	
GetOnSked	
GetUp	
HelpedUp	
Idle	
Prone	
RolloverProne	
Run	
Supine	
SupineForSked	

TalkOnBack1	
TalkOnBack2	
TalkOnBack3	
TurnLeft	
TurnRight	
Twitch	
Vomit	
WalkBackwards	
WalkForward	
WalkLeft	
WalkRight	
Wander	

Terrorist01

Terrorist 01

The Terrorist Actor consists of geometry and animations. The terrorist actor is largely the same as the victims with the exception that the terrorist has a gun in his left hand.

<Property>	<Description>
None	

<Animation>	<Description>
Choke	
Convulse	
Cough	
Die	
FallDown	
GetOnSked	
GetUp	
HelpedUp	
Idle	

Prone	
RolloverProne	
Run	
Supine	
SupineForSked	
TalkOnBack1	
TalkOnBack2	
TalkOnBack3	
TurnLeft	
TurnRight	
Twitch	
Vomit	
WalkBackwards	
WalkForward	
WalkLeft	
WalkRight	
Wander	

Emergency Responder

Emergency Responder

The Emergency Responder consists of the geometry of the actor and animation. Scenario 2 and 3 will use the Emergency Responder actor to depict the members of the ERT.The members of the team will have the following roles.

- ***HazMat Recon Team Leader.*** The HazMat Recon Team Leader will be wearing Level "A" Protective Clothing and a two-way radio. The HazMat Recon Team Leader responsibilities are appearance are similar to the Survey Team Leader of Scenario1.
    - Supervise deployment and operation of all monitoring, detection and sampling equipment
    - Direct all movement of the HazMat Survey Team in the hot zone.
    - Maintain visual accountability of all team members while in the hot zone at all times.
    - Generate team log while down range. Log will include at a minimum:
        - Sector sketches (photo or video)
        - Background / Pre-entry equipment readings
        - Time, Location and type of sample
        - Be prepared to direct rescue operations in the hot zone.

- ***HazMat Recon Team Member.*** The HazMat Recon Team member will be wearing Level "A" Protective Clothing and a two-way radio. The HazMat Recon Team Member responsibilities and appearance are similar to the Assistant Sampler of Scenario1:
    - Receive pre/post-entry medical exam.
    - Attend mission briefs and ask pertinent questions to clarify mission.

- Identify and inform Team Leader of any and all additional equipment needs.
- Perform all tasks in the hot zone as directed by the Team Leader

Each ERT member has with him a collection of equipment that he must utilize to successfully complete the scenario. The following is a list of that equipment.

- ***Two-way Radio.*** Allows team members to communicate with the HazMat Section Officer and incident command.
- ***Level "A" PPE.*** This personal protection equipment (PPE) provides the responder with the highest protection against unknown respiratory, liquid and vapor chemical hazards. The enclosed self-contained breathing apparatus (SCBA) will provide up to one-hour or breathing time for the responder.
- ***Thermal-Imaging Camera (TIC)***. This provides the responder with the capability to scan the Hot Zone and detect the presence of WMD agents or hazards not visible to the normal eye.
- ***APD2000.*** This is a portable WMD hazmat detection & ID instrument that can detect the presence of a "G" Nerve, "H" agent.
- ***Bioseeq.*** Handheld bio agent detector.
- ***Immuno Assay Tickets.*** Measures for the presence of biological agents.
- ***M256A2 Kit***. This is a chemical warfare agent detection card that can detect the presence of both liquid and vapor nerve, blister, cyanide and choking chemical warfare agents in a tactical environment
- ***MultiRae.*** This is a portable instrument that can detect and measure the oxygen percentage, upper explosive limit (UEL), lower explosive limit (LEL), carbon monoxide and hydrogen sulfide in a tactical environment.
- ***Sabre4000***. Handheld chemical and explosive agent detector.
- ***GasID.*** Portable gas and vapor identifier.
- ***HazmatID.*** Infrared chemical identifier.
- ***Triage Tag.*** Each Special Triage and Rapid Treatment (START) triage tag gives the responder the capability to visually classify a specific patient as to the priority of emergency care needed based on observed and manually extracted medical signs and symptoms. This is achieved by writing on the waterproof tag with a grease pencil and tearing off the tabs to achieve the correct triage color designation.
- ***Atropine Auto-Injector.*** This is a 2mg auto-injector as part of a Mark 1 antidote kit which stops the effects of nerve agent poisoning and is designed to be administered through clothing in a tactical environment. It is a spring loaded syringe that releases the dosage when it is pressed against a clothing surface via a 1-1/2 needle.
- ***2-PAM Auto-Injector.*** This is a 2mg auto-injector as part of a Mark 1 antidote kit which reactivates normal nerve end messaging and is designed to be administered through

clothing in a tactical environment. It is a spring loaded syringe that releases the dosage when it is pressed against a clothing surface via a 1-1/2 needle.

- ***NAAK Auto-Injector***. This is a 10mg auto-injector containing Valium to counteract the life-threatening of seizures encountered as part of chemical poisoning and is designed to be administered through clothing in a tactical environment. It is a spring loaded syringe that releases the dosage when it is pressed against a clothing surface via a 1-1/2 needle.
- ***B.A.L. Auto-Injector***. This is a 0.5mg/25lb British Antidote Lewisite (BAL) drug injector is designed to counter the painful effects of contamination from Lewisite blister agent. It is a spring loaded syringe that releases the dosage when it is pressed against a clothing surface via a 1-1/2 needle.
- ***Cyanide Antidote Kit***. This drug kit contains Amyl Nitrate for use in countering the life-threatening effects of high concentrations of cyanide warfare agent. The ampoule must be crushed under the patient's nose in order to release its dosage.
- ***SKED Stretcher***. This is a stretcher especially designed to allow a rapid extrication of a non-walking victim to the decontamination area. The SKED allows the medic to rapidly roll the victim onto the stretcher, bring up the side, foot, and head portions, buckle the belts and drag the SKED which easily moves across rough terrain.

<Property>	<Description>
None	

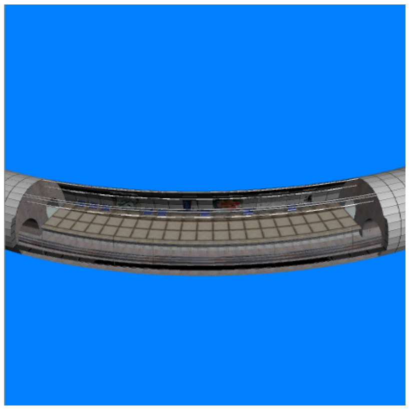
<Animation>	<Description>
FallDown	
GiveBothHands	
GiveLeftHand	
GiveRightHand	
HelpVictim	
Idle	
LoadOntoSked	
LowerDeviceRightHand	
PointForward	
PointLeft	

PointRight	
Prone	
PushCartIdle	
PushCartWalkBackward	
PushCartWalkForward	
PushCartWalkLeft	
PushCartWalkRight	
PushCartWalkTurnLeft	
PushCartWalkTurnRight	
RaiseDeviceRightHand	
Supine	
SupineForSked	
Talk	
TurnLeft	
TurnRight	
UseAutoInjector	
UseDeviceBothHands	
UseDeviceBothHandsIdle	
UseDeviceBothHandsWalkBackward	
UseDeviceBothHandsWalkForward	
UseDeviceBothHandsWalkLeft	
UseDeviceBothHandsWalkRight	
UseDeviceBothHandsWalkTurnLeft	
UseDeviceBothHandsWalkTurnRight	
UseDeviceLeftHand	

UseDeviceLeftHandBent	
UseDeviceLeftHandIdle	
UseDeviceLeftHandWalkBackward	
UseDeviceLeftHandWalkForward	
UseDeviceLeftHandWalkLeft	
UseDeviceLeftHandWalkRight	
UseDeviceLeftHandWalkTurnLeft	
UseDeviceLeftHandWalkTurnRight	
UseDeviceLeftWrist	
UseDeviceRightHand	
UseDeviceRightHandBent	
UseDeviceRightHandIdle	
UseDeviceRightHandWalkBackward	
UseDeviceRightHandWalkForward	
UseDeviceRightHandWalkLeft	
UseDeviceRightHandWalkRight	
UseDeviceRightHandWalkTurnLeft	
UseDeviceRightHandWalkTurnRight	
UseDeviceRightWrist	
UseTriageTag	
WalkBackward	
WalkForward	
WalkLeft	
WalkRight	
WalkWithKitBack	

WalkWithKitForward	
WalkWithKitLeft	
WalkWithKitRight	
WalkWithSked	
Die	
CrouchHigh	
CrouchLow	

Subway Station

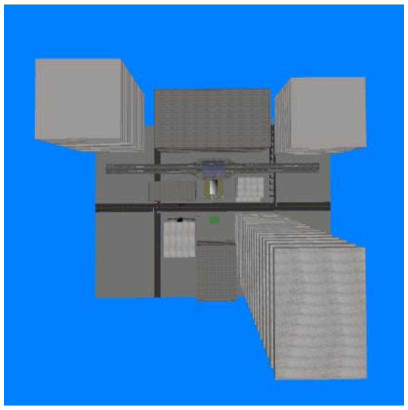

Subway Station

The Subway Station Actor consists only of the geometry of the Subway Station. The Subway Station is one of two settings for Scenarios 1 and 2. The Subway Station has two exits at either end of the platform that lead up to the Train Station.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Train Station


Train Station

The Subway Station Actor consists only of the geometry of the Train Station. The Train Station is located in the middle of a city. The platform and waiting area are the setting for Scenarios 1 and 2. Stairwells on the train platforms lead down to the Subway Station.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Clipboard

The clipboard actor provides GUI for depositing markers from the legend onto a map.

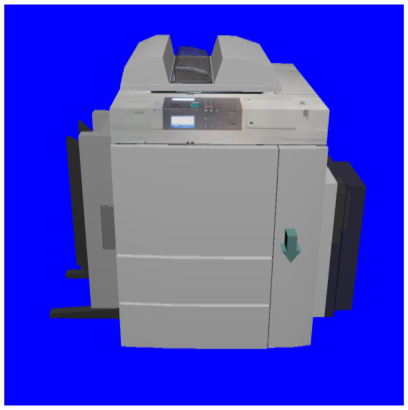
<Property>	<Description>
`Background.Anchor`	The anchor of the background.
`Background.Size`	The background size.
`Background.Color`	The background color.
`Texture`	The texture used for the background.
`ActiveOverlay.Visible`	The visibility of the active overlay.
`ActiveOverlay.Size`	The size of the active overlay.
`ActiveOverlay.Position`	The position of the active overlay.
`ActiveOverlay.Color`	The active overlays color.
`ActiveOverlay.Texture`	The texture used for the active overlay.
`NumberOfOverlays`	The number of current overlays.
`Overlays`	Collection of overlays.
`Legend.Alignment`	The alignment of the legend.
`Legend.Span`	The span of the legend.
`Legend.Margin`	The size of the legends margin.
`Legend.Color`	The background color of the texture.
`Legend.Texture`	The texture displayed behind the legend.
`Legend.Font`	The font used for the legend’s text.
`Legend.FontColor`	The color of the font used in the margin.
`Legend.HighlightMargin`	The margin on a highlight.

`Legend.HighlightColor`	Highlight color when a marker is selected in the legend.
`Legend.Title`	Title for the legend.
`Legend.TitleMargin`	Margin for legend’s title.
`Legend.TitleFont`	Font to use for displaying the legend’s title.
`Legend.TitleColor`	Color of font to use for displaying the legend’s title.
`NumberofMarkerTypes`	The number of different marker types available.
`MarkerTypes`	The list of marker types.
`Markers`	The current list of markers.

COPIER COMPONENTS

Copier Components are found in the <Install>.Components\Copier directory.

Copier



Copier

The Copier consists of only the geometry of the copier. All animations are done procedurally on a per part basis. Each movable part is represented as a separate joint.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Copier Status

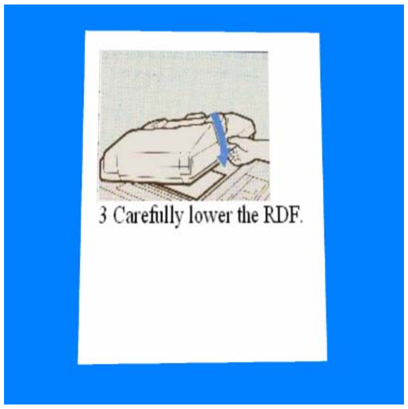
Copier Status

The Copier Status consists of only the plane geometry onto which the LCD texture is mapped.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Copier HUD



Copier HUD

The Copier HUD consists of only the plane geometry onto which the documentation is mapped.

<Property>	<Description>
None	

<Animation>	<Description>
None	

Office

Office

The Office actor consists only of the geometry representing an office environment.

<Property>	<Description>
None	

<Animation>	<Description>
None	